KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

FINITE DOMAIN AND SYMBOLIC INFERENCE METHODS FOR EXTENSIONS OF FIRST-ORDER LOGIC

Johan WITTOCX

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering

May 2010

FINITE DOMAIN AND SYMBOLIC INFERENCE METHODS FOR EXTENSIONS OF FIRST-ORDER LOGIC

Johan WITTOCX

Jury:
Prof. Dr. ir. Willy Sansen, president
Prof. Dr. Marc Denecker, promotor
Prof. Dr. ir. Maurice Bruynooghe
Prof. Dr. Jan Denef
Prof. Dr. ir. Joos Vandewalle
Prof. Dr. Mirosław Truszczyński
(University of Kentucky)
Prof. Dr. Torsten Schaub
(Universität Potsdam)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

May 2010

D/2010/7515/35
ISBN 978-94-6018-196-2

Abstract

The scientific field of *Knowledge Representation and Reasoning* (KRR) is concerned with the development of formal languages (logics) to express knowledge and with inference methods to extract new knowledge. One of the long term goals of KRR is to build a knowledge base system (KBS). In such a system, knowledge about a domain of discourse is stored and different tasks are solved by applying various inference methods on that knowledge. In this thesis, we investigate a rich extension of classical logic, suitable as underlying language for a KBS, and we look at various sorts of inference for this logic in finite contexts.
The basic inference we examine is *constraint propagation*, the task of deriving from a logic theory facts that are certainly true in every model of the theory, i.e., in every situation that is possible according to the theory. We mainly investigate an incomplete but efficient form of constraint propagation, which has the benefit that it can be executed in a symbolic manner. We indicate several applications of constraint propagation.
The second form of inference we study in this thesis is *grounding*, the task of transforming a logic theory containing variables into an equivalent propositional theory, i.e., a theory without variables. Grounding is frequently used as a first phase in systems for finite model generation, another important inference task. We show how grounding can be improved by adding redundant information to a theory. This redundant information can be computed by applying symbolic constraint propagation.
Next, we study the task of *debugging* a logic theory. The debugging method we propose consists of interactively examining formal proofs of the inconsistency of a theory. We show that model generators based on constraint propagation can automatically generate the formal proofs.
Finally, we investigate *model revision*, i.e., adapting a model of a theory in case new requirements are added. Model revision has applications in, for example, reconfiguration and rescheduling. We present a basic method to solve model revision problems by solving sequences of model generation problems.

Samenvatting

Het wetenschappelijk onderzoeksdomein *Kennisrepresentatie en redeneren* beoogt het ontwikkelen van formele talen (logica's) die geschikt zijn om kennis uit te drukken en van inferentiemethodes om te redeneren over kennis uitgedrukt in die talen. Een van de hoofddoelen binnen dit onderzoeksdomein is het bouwen van een kennisbanksysteem (KBS): een systeem waarin een menselijke expert zijn kennis over een bepaald domein opslaat en waarmee verschillende taken in dat domein opgelost worden door het toepassen van inferentiemethodes. In deze thesis stellen we een uitbreiding van klassieke logica voor als geschikte logica voor een KBS en onderzoeken we verschillende vormen van inferentie.
De eerste vorm van inferentie die we onderzoeken is *propagatie*: uit een logische theorie feiten afleiden die zeker waar zijn in elk model van de theorie, dat wil zeggen in elke situatie die mogelijk is volgens de theorie. We onderzoeken voornamelijk een benaderende maar efficiënte vorm van propagatie, die bovendien op een symbolische manier uitgevoerd kan worden. We beschrijven verschillende toepassingen van propagatie.
De tweede vorm van inferentie is *propositionalisatie*: het omzetten van een logische theorie die variabelen mag bevatten naar een equivalente propositionele theorie. Propositionalisatie wordt vaak gebruikt als eerste stap in systemen voor eindige model generatie. Eindige model generatie vormt op zich ook een belangrijke vorm van inferentie. We tonen aan hoe propositionalisatie verbeterd kan worden door overtollige informatie toe te voegen aan een theorie. Deze overtollige informatie kan berekend worden met behulp van symbolische propagatie.
Ten derde bestuderen we hoe fouten in een logische theorie opgespoord kunnen worden. De methode die we voorstellen bestaat uit het interactief overlopen van formele bewijzen van de inconsistentie van een theorie. We tonen dat model generatoren die gebaseerd zijn op propagatie gebruikt kunnen worden om automatisch zulke formele bewijzen op te stellen.
Tenslotte bestuderen we *model revisie*: het aanpassen van een model van een theorie wanneer nieuwe vereisten gesteld worden. Model revisie heeft onder andere toepassingen in herconfiguratie en herplanningsproblemen. We laten zien hoe model revisie problemen aangepakt kunnen worden door het oplossen van opeenvolgende model generatie problemen.

Acknowledgments

In one of the books that have impressed me in recent years it is argued that: "To see reality in truth, we must see it in relation to God". The reason for this is that "God created reality, sustains it, and gives it all the properties it has and all its relations and designs" (Piper, 2000). One may wonder if the same applies to research areas such as Mathematics and Computer Science. Are they not the study of mere human constructs? These acknowledgments are not the place to give a detailed answer to this question, but in *Artificial Intelligence* and in *Knowledge Representation and Reasoning*, the areas of Computer Science where this thesis belongs, the answer, in my opinion, is clearly given. The very fact that we, as human beings can gather knowledge, can store it and can reason about it, is, in my view, one of the incredible and intriguing aspects of God's creation. For this reason I first of all wish to thank God; in general for the ability He has given man to do scientific research, and in particular for the *non*-artificial intelligence He has placed in Creation. This intelligence is so complex that even after more than fifty years of studying it and mimicking it, there is still ample room for doctoral theses such as this.

There are also numerous *people* to whom I am thankful for their contribution to my studies on my journey between graduation from secondary school and earning my PhD. Danny van Doninck stirred up my interest in real mathematics — in contrast to arithmetic — and encouraged me to study for a Masters Degree in Mathematics. During this training I very much enjoyed the courses on Logic and Number Theory given by Jan Denef. It was also Jan who pointed out to me the possibility of doing a doctorate in the area of Computer Science. With the help of Bart Demoen and Maurice Bruynooghe I came into contact with Marc Denecker, who became my promoter.

Marc had the difficult task of transforming a self-critical theoretician and fanatical mathematician into someone capable of writing legibly concerning Computer Science research. The fact that the chapters of this thesis start off with motivational introductions instead of "definition 1.1.", the fact that a relatively large amount of illustrations — even real-life examples — are given and the fact that not only the negative aspects of experimental results are highlighted, show that Marc has succeeded in this task.[1] Apart from writing style, Marc also had

[1] That the 'List of Symbols' extends beyond more than three pages, is entirely my own responsibility.

great influence on the content of my thesis. Many of the ideas presented in this work originated during discussions and brainstorming sessions with him which invariably followed the codeword "coffee?", usually around 10 am. I am also indebted to Marc for the nice study trips to Vancouver and Sydney and for his — mostly completely unexpected — questions on challenging ethical issues.
I also thank the other members of my jury for their willingness to read through this text and for their useful comments. Furthermore, I would like to thank Maurice Bruynooghe for his help with administrative and financial affairs, Jan Denef for his never failing enthusiasm for my research, Mirek Truszczyński for asking a not-too-difficult question following my very first international presentation (at ICLP 2006) on a subject I still did not know much about back then, Torsten Schaub for the opportunity to stay in Potsdam for two weeks in order to exchange ideas with his research group, Joos Vandewalle for his willingness to join the jury just one month before the public defence, and finally Willy Sansen for his willingness to preside on the jury.
I thank my colleagues Maarten, Joost, Stephen, Alvaro, Hou, Broes, Hanne, Stef, Abhinav, Mai and Alexandre of the KRR group for the non-productive but pleasant lunch, coffee and table tennis breaks, barbecues and "paper celebrations". I especially thank Maarten for the good collaboration, the nice trips, for his fast and excellent feedback on my texts and for answering my questions on Linux and C++. Alvaro and Hou: Thank you for being my office mates for such a long time, in what was probably the quietest KRR office. Broes: Thank you for proving that it is possible to successfully finalise a Master's thesis under my supervision.
The research of this thesis would not have been possible without the financial support of the Research Foundation Flanders (FWO), initially via the projects G.0202.02 and G.0357.06, and later via a personal grant.
Besides those who directly contributed to this thesis, I also want to thank those who helped to make the past five years relaxed, pleasant and varied. My gratitude goes to my friends from faraway Mechelen, from Ichtus Leuven, from International Baptist Church, and from the Sven Salamander and Blauwput table tennis clubs. I thank my parents and parents-in-law, brothers, sister and sister-in-law, for their support and friendship. And finally I thank Joke and Natanaël. They made it a joy to come home in the evenings.

Johan Wittocx
Leuven, May 2010

Contents

List of Symbols

$\tilde{\Phi}$ A four-valued symbolic structure, page 69

$\tilde{I}[P(\overline{d})/\mathbf{v}]$ The structure that assigns $\mathbf{v}$ to $P(\overline{d})$ and corresponds to $\tilde{I}$ on all other symbols, page 20

atom(L) The domain atom associated to the domain literal L, page 20

$\mathfrak{base}$ A function mapping sorts to their base sort., page 32

$\mathcal{C}$ A c-map, page 105

$\overline{\mathcal{C}}$ The theory associated to c-map $\mathcal{C}$, page 106

$\overline{\mathcal{C}}_A$ The sentences form $\overline{\mathcal{C}}$ associated to atomic formulas, page 109

$\mathcal{C}\langle\varphi\rangle$ The c-transformation of φ with respect to $\mathcal{C}$, page 107

$\mathcal{D}_S(\varphi)$ The set of dangerous literals in φ with respect to S, page 155

$\mathscr{I}(V)$ The set of INF propagators associated to the INF sentences in V, page 59

$\mathscr{I}^{\varphi}$ The implicational normal form propagator associated to φ, page 58

INF(φ) The set of INF sentences associated to ENF sentence φ, page 63

I_σ An input structure for a model expansion problem, page 101

$\mathfrak{Int}$ The sort of all integers, page 35

$\rightarrowtriangle$ The if-then-else connective, page 76

$\mathcal{T}$ An MX-tree, page 140

$\models_{I_\sigma}$ Satisfaction relation with respect to an input structure I_σ, page 101

$\mathrm{Gr}_{\mathrm{full}}(T)$ The full grounding of theory T, page 103

$\ominus$ The negative sign, page 137

$\oplus$ The positive sign, page 137

Φ A symbolic two-valued structure, page 68

$\varphi^{\oplus}, \varphi^{\ominus}$ Signed instances, page 137

$\mathrm{Gr}_{\mathrm{red}}(T)$ The reduced grounding of theory T, page 104

$\mathscr{I}_s^{\varphi}$ The symbolic INF propagator associated to φ, page 71

$|\overline{a}|$ The number of elements in tuple $\overline{a}$, page 12

s_{pred} The sort predicate of sort s, page 32

$\mathrm{swap}(I, R)$ The structure obtained from I by swapping the truth value of atoms in R, page 152

$\mathscr{L}_T(\tilde{I})$ The limit of the terminal T-refinement sequence from $\tilde{I}$, page 56

$\varphi[t/t']$ The formula obtained by replacing t by t' in φ, page 12

$\Sigma^{\mathrm{dom}(\tilde{I})}$ The extension of vocabulary Σ with domain constants, page 59

P^{Φ} The query assigned to P by symbolic structure Φ, page 68

S A symbolic propagator, page 70

$v \sqsubset \tilde{V}$ v is a set approximated by the three-valued set $\tilde{V}$, page 46

$\tilde{I}^{\mathrm{cf}}$ The cf-part of structure $\tilde{I}$, page 23

$\tilde{I}^{\mathrm{tf}}$ The two-valued Σ^{tf}-structure associated to $\tilde{I}$, page 24

$\tilde{I}^{\mathrm{ct}}$ The ct-part of structure $\tilde{I}$, page 23

$\leftarrow$ Definitional implication, page 39

$\mathrm{Def}(\Delta)$ The defined symbols of definition Δ, page 39

$\Delta_{\tilde{I}}$ The definition associated to structure $\tilde{I}$, page 60

$\mathrm{Open}(\Delta)$ The set of symbols that are open with respect to definition Δ, page 39

Δ_V The definition associated to the set V of INF sentences, page 60

$\Delta_{V,\tilde{I}}$ The union of Δ_V and $\Delta_{\tilde{I}}$, page 60

$|\tilde{I}|$ The size of the domains of structure $\tilde{I}$, page 20

$\|\tilde{I}\|$ The size of $\tilde{I}$, page 20

FO(PF) FO extended with partial functions, page 28

$F, G, \ldots$ Function symbols, page 9

$\mathscr{G}_F$ A predicate symbol denoting the graph of the function denoted by F, page 18

$\theta[\overline{x}/\overline{d}]$ The variable assignment that assigns $\overline{d}$ to $\overline{x}$ and corresponds to θ on all other variables, page 12

Σ A vocabulary, page 9

Σ_{func} The set of function symbols of vocabulary Σ, page 9

Σ_{pred} The set of predicate symbols of vocabulary Σ, page 9

Σ_{sort} The set of sorts of vocabulary Σ, page 9

Σ_{base} The base sorts of vocabulary Σ, page 32

Σ_{sub} The subsorts of vocabulary Σ, page 32

Σ^{tf} The vocabulary that contains symbols P^{ct} and P^{cf} for each symbol P of Σ, page 24

Σ_{var} The variables of vocabulary Σ, page 9

$\text{wfm}_{\Delta}(\tilde{I})$ The well-founded model of definition Δ extending $I|_{\text{Open}(\Delta)}$, page 40

Chapter 1

Introduction

The field of Knowledge Representation and Reasoning (KRR) is concerned with the development of formal languages to express knowledge — commonly called *logics* — and of inference methods to reason about knowledge expressed in these languages. A well-known example of a logic is *classical first-order logic* (FO). In FO, statements such as "All humans are mortal" and "Socrates is a human" are expressed by, e.g.,

$$\forall x\ (Human(x) \Rightarrow Mortal(x)); \tag{1.1}$$

$$Human(Socrates). \tag{1.2}$$

An example of an inference method is *deduction*. It helps to derive new knowledge from a given set of statements. For instance, we could use deduction to derive from statements (1.1) and (1.2) that Socrates is mortal. Another example of an inference method is *model generation*, the task of constructing a situation that is possible according to a theory.
A long-term goal of KRR is the development of a *knowledge base system* (KBS). In a KBS, an expert stores his knowledge about a certain domain of discourse and solves different tasks in that domain by applying appropriate inference methods on that knowledge. An example is a KBS storing knowledge about course scheduling at a university. The knowledge in this KBS contains statements like

$$\begin{aligned}\forall l_1 \forall l_2\ (&Lecture(l_1) \wedge Lecture(l_2) \wedge l_1 \neq l_2 \Rightarrow \\ &\neg(Time(l_1) = Time(l_2) \wedge Place(l_1) = Place(l_2))),\end{aligned} \tag{1.3}$$

expressing that two different lectures cannot be scheduled at the same time and place. By applying suitable forms of inference, schedules can be generated automatically at the start of the year, hand-made schedules can be checked, existing schedules can be revised, etc., all using the same knowledge.
There are two important prerequisites to make a KBS applicable in practice. First of all, the logic underlying the KBS should allow to represent a wide variety of knowledge in a precise, intuitive and concise manner. Technically speaking,

it should have a high expressivity and a clear formal and informal semantics. The second requirement is that there exist reasonably cheap, but nevertheless useful inference methods for the KBS language. In this thesis, we propose a rich extension of FO as KBS language and we investigate several inference methods for this logic.

1.1 Extending first-order logic

Denecker and Vennekens (2008) argued that FO meets the criteria for being the basis of a suitable KBS language. Indeed, it has a clear formal and informal semantics and has been used to represent knowledge in several domains (see, e.g., the theories in the TPTP library, Sutcliffe, 2009). Moreover, it is by far the most studied logic. However, there are several concepts that occur frequently in real-life applications but cannot be expressed in FO. A well-known example is the definition of *reachability*: the sentence "There is a path from position A to position B" cannot be formulated in FO using a vocabulary that can only refer to direct connections between positions. Another example is the aggregate function mapping a set of integers to the sum of its elements. To overcome these limitations, several extensions to FO have been proposed in the literature. For example, FO(ID) (Denecker and Ternovska, 2008) extends FO with a construct to represent inductive definitions. Reachability can easily be expressed in FO(ID).

Some additions to FO are not introduced to extend its expressivity, but rather to allow for more concise and readable theories. An example is *many-sorted* FO. This logic is useful for classifying all objects of a domain into different *sorts* (or *types*). Many-sorted FO can be simulated in FO, but this leads to larger and more error-prone theories.

Still other extensions proposed in the literature are added primarily to improve the efficiency of a certain inference task. Many constraints in input languages of constraint solvers serve this goal. In the context a KBS, such extensions are of less importance since it is precisely the aim to apply different forms of inference using the same theory. A specific way of writing a certain piece of knowledge may improve the efficiency of an algorithm implementing one form of inference, but may at the same time harm the efficiency of an algorithm implementing another form.[2]

In this thesis, we present the logic FO($\cdot$), a non-hybrid combination of several of the existing extensions to FO, namely partial functions, sorts, integer arithmetic, aggregates and inductive definitions. As such, FO($\cdot$) is a good candidate to serve as a KBS language. In fact, FO($\cdot$) is a moving target: at the moment, several extensions of FO($\cdot$) are being investigated in our research group. The version presented in this thesis is the one currently implemented in IDP, a system for generating finite models.

[2] An interesting research topic is how to automatically rewrite theories such that they become "suitable" for a certain form of inference. In Chapter 5, we investigate this for grounding.

1.2 Forms of inference

As mentioned above, the logic used in a KBS should allow for several reasonably cheap, but useful sorts of inference. However, the more expressive a logic, the more expensive inference for that logic becomes. For instance, *deduction* is a useful form of inference, as witnessed by the applications of description logics (see, e.g., Baader et al., 2003). Yet deduction for FO is undecidable. As such, a KBS using (an extension of) FO as language cannot implement deduction as a reliable sort of inference.

Nevertheless, it has become apparent over the last decade that even for a logic as expressive as FO($\cdot$), several useful and reasonably cheap sorts of inference exist. For example, Marek and Truszczyński (1999) pointed out the use of *finite model generation* for solving many computational problems. Mitchell and Ternovska (2005) indicated that *finite model expansion*, an extension of model generation, can be applied to solve all problems in **NP**. In some expressive description logics, efficient *approximate* forms of inference are being investigated (Stuckenschmidt, 2007). Wittocx et al. (2008a) developed a useful form of symbolic reasoning for FO.

In this thesis, we investigate several forms of inference for FO($\cdot$) and describe different applications. As a first form of inference, we examine *constraint propagation* for FO($\cdot$): the task of deriving from a given theory and set of facts, new facts that are implied by the theory. The set of facts is given as, e.g., a database. For example, if the theory contains sentence (1.3) and it is given that the first lecture of the course *Object-oriented programming* is scheduled from 10am till 12am in room A0.125, then constraint propagation derives that no other lecture is scheduled at that time slot in room A0.125.

In many applications, it is important that constraint propagation is at least tractable. For FO and FO($\cdot$) however, *complete* constraint propagation, i.e., deriving all implied facts, turns out to be intractable. We therefore investigate an incomplete but efficient propagation method. One of the benefits of our method is that it allows us to represent the propagation as a monotone inductive definition, defining which facts are derived. Several advanced techniques to efficiently evaluate such definitions have been studied in the literature. These techniques can be applied to implement our propagation method. Another benefit of our method is that it can be applied in a symbolic way, i.e., independent of the given facts. The symbolic method may, e.g., derive from (1.1) and (1.2) that any x that is equal to the person denoted by *Sokrates*, is mortal. We show that our (symbolic) propagation method has applications in approximate query answering in incomplete databases, in finite model generation and in building configuration systems.

The second inference task we study is *grounding*: transforming a first-order theory, i.e., a theory containing variables, and finite domain into an "equivalent" propositional theory. As such, grounding serves to solve problems on the first-order level by applying systems for reasoning on propositional theories. Grounding is frequently used in systems for finite model generation, such as *answer set solvers*. A common practice to improve grounding consists of man-

ually adding redundant information to the first-order input theory. We present a method to automate this process. We show where redundant information can be added, and how redundant information can be obtained by applying our symbolic constraint propagation method. Based on this method, we developed an FO(·) grounding algorithm and implemented it in the grounder GIDL. GIDL is part of the FO(·) finite model generation system IDP, developed in close collaboration with Maarten Mariën.

We study two other inference tasks in less detail. The first one concerns *debugging* theories, a vital task in a real-life KBS system. Although FO has a clear formal and informal semantics, mistakes are made when expressing knowledge in a concrete FO theory. Such bugs may have a serious impact and are often difficult to detect. For example, a common mistake is to assume that different variables take different values, resulting in writing

$$\begin{aligned} \forall l_1 \forall l_2 \ (&Lecture(l_1) \wedge Lecture(l_2) \Rightarrow \\ &\neg(Time(l_1) = Time(l_2) \wedge Place(l_1) = Place(l_2))), \end{aligned} \tag{1.4}$$

instead of sentence (1.3). According to sentence (1.4), no lecture can be scheduled at all. We develop a debugging method that consists of exploring formal proofs. In the example, such a proof explains why no lecture can be scheduled. We show that these formal proofs can be constructed using a finite model generator based on constraint propagation.

The second form of inference studied in less detail is finite *model revision*. Here, the problem is that of adapting a model of a theory to changed circumstances. Applications can be found in, e.g., network configuration. Such a configuration can be represented as a model of a theory describing constraints on configurations. When a server breaks down, it is possible that the network needs to be reconfigured, i.e., a new model of the theory needs to be computed. Often, it is not needed that the new model is computed from scratch. Instead, most of the time, small and local modifications to the previous model are sufficient. We present a method, based on grounding and propositional model generation, to find such local modifications.

In this thesis, constraint propagation and grounding are developed for full FO(·). Debugging is investigated for FO(ID), and model revision for FO. Extending the results about debugging to full FO(·) is straightforward. For model revision, this is much more complicated. It is part of future work to extend our results about model revision to full FO(·).

1.3 Structure of the text

The rest of this text is structured as follows:

- In Chapter 2 we introduce the technical background and notations used in the rest of the text. Specifically, the definitions of many-sorted first-order logic and four-valued structures are recalled.

- In Chapter 3 we present FO(·). We include references to the papers that introduced each of the individual extensions contained in FO(·).
- Constraint propagation is investigated in Chapter 4. Large parts of this chapter were published in (Wittocx et al., 2008a). First, some general results about propagation are presented. Next, we develop a polynomial constraint propagation method for FO theories. The symbolic variant of this method is introduced in Section 4.3. Then, we extend our results to FO(·). Finally, three applications are discussed.
- Grounding is introduced in Chapter 5. Large parts of this chapter were published in (Wittocx et al., 2008c, 2010). Section 5.2 explains how redundant information can be added to FO theories in order to improve grounding efficiency. Section 5.3 lifts the results to FO(·). A concrete grounding algorithm is presented in Section 5.4. We show by experiments the impact of our method on grounding time and size. We end with a discussion of related work.
- Chapter 6 presents our debugging method. First the formal proof calculus is introduced. Next, we show how the formal proofs can be examined interactively. The content of this chapter was first published in (Wittocx et al., 2009b).
- Model revision is introduced in Chapter 7. Several variants of model revision are discussed, followed by a basic algorithm to solve model revision problems. We report on a prototype implementation. The content of this chapter is joint work with Broes De Cat and was published in (Wittocx et al., 2009a). Broes' (2009) master's thesis elaborates on some of the topics.
- Chapter 8 lists our conclusions.

As to not interrupt the flow of the text, the proofs of many of the theorems are given in the appendix, rather than directly below statement of the theorem itself. This was done primarily for long but easy proofs that do not provide important additional insight. Also some more advanced topics are covered in the appendix, as well as some technical details about the concrete input syntax and implementation of GIDL.

Chapter 2

Preliminaries

In this chapter we introduce the technical background and the notations used throughout this thesis. We first review first-order logic (FO), also called *classical logic*. Then we present three- and four-valued structures. We will illustrate many of the concepts we introduce by means of the following example.

Example 2.1 (Battleship puzzle)**.** A *battleship puzzle* consists of a grid and a number of ships of different lengths. Each row and column in the grid has an associated number. An example puzzle is shown in Figure 2.1. The goal is to place the ships in the grid in such a position that they do not touch each other (not even diagonally) and that for each row and column the number of occupied squares on that row, respectively column, matches exactly the associated number. A partial solution may be given to guide the puzzler. For example, in Figure 2.1, it is indicated that square $(2, 3)$ is occupied by part of a ship with length at least two, and that square $(9, 5)$ contains no ship. A solution to the puzzle in Figure 2.1 is shown in Figure 2.2.

We assume that the reader is familiar with the basic notions and notations of set theory. We refer to Enderton's (1977) book for a good introduction to set theory. We also assume a basic knowledge about computability and computational complexity (see, e.g., Sipser, 2005).

2.1 First-order logic

Like any language, a *formal language* has a vocabulary (the symbols and words that can be used to build sentences) and a syntax (the rules to build correct sentences). In formal languages the vocabulary and syntax are described in a mathematically precise manner. Besides a vocabulary and a syntax, many formal languages also have formal semantics. That is, there is a mathematically precise description of the meaning of the sentences in the language. Such languages are called *logics*. In this section, we review the vocabulary, syntax and semantics of *first-order logic* languages. To describe the semantics, we need to

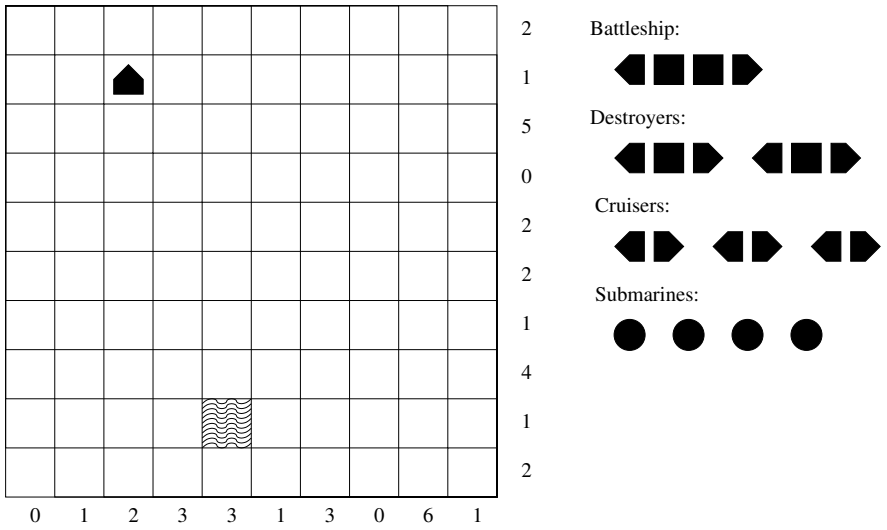


Figure 2.1: A battleship puzzle

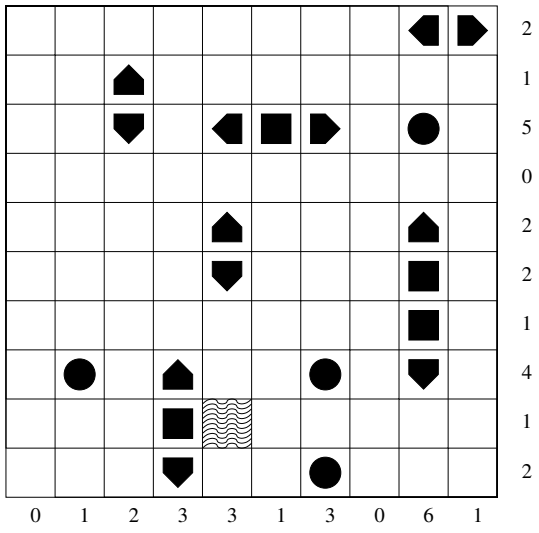


Figure 2.2: A solved battleship puzzle

introduce the concept of a *structure*. A structure can be seen as a description of a state of (part of) the world. Given a structure, the formal semantics of a first-order logic language expresses which sentences of the language are true in the structure.

2.1.1 Vocabulary, syntax and semantics

We immediately introduce the many-sorted version of FO. Afterwards, we also explain the standard one-sorted version.

Vocabulary

A *vocabulary* Σ consists of a set Σ_{sort} of *sorts*, a set Σ_{pred} of *predicate symbols*, a set Σ_{func} of *function symbols* and a set Σ_{var} of *variables*. We sometimes introduce a specific vocabulary Σ by the expression $\Sigma = \langle V_s, V_p, V_f, V_v \rangle$. This denotes that Σ is the vocabulary defined by $\Sigma_{\text{sort}} = V_s$, $\Sigma_{\text{pred}} = V_p$, $\Sigma_{\text{func}} = V_f$ and $\Sigma_{\text{var}} = V_v$. Abusing notation, we often omit V_v and simply write $\Sigma = \langle V_s, V_p, V_f \rangle$. A vocabulary Σ' is a *subvocabulary* of Σ, denoted $\Sigma' \subseteq \Sigma$, if $\Sigma'_{\text{sort}} = \Sigma_{\text{sort}}$, $\Sigma'_{\text{pred}} \subseteq \Sigma_{\text{pred}}$, $\Sigma'_{\text{func}} \subseteq \Sigma_{\text{func}}$ and $\Sigma'_{\text{var}} \subseteq \Sigma_{\text{var}}$.[3]
Each predicate symbol from Σ has an associated number of arguments, called its *arity*. Each predicate symbol $P \in \Sigma_{\text{pred}}$ with arity n has an associated tuple $(s_1, \ldots, s_n)$ of sorts from Σ_{sort}. We denote this tuple by $\mathfrak{s}(P)$. Likewise, each function symbol $F \in \Sigma_{\text{func}}$ has an associated arity n and tuple of sorts $(s_1, \ldots, s_{n+1})$, denoted by $\mathfrak{s}(F)$. A variable $v \in \Sigma_{\text{var}}$ has an associated sort $\mathfrak{s}(v) \in \Sigma_{\text{sort}}$. We often denote a predicate symbol P by P/n or by $P(s_1, \ldots, s_n)$ to indicate, respectively, that P has arity n or that $\mathfrak{s}(P) = (s_1, \ldots, s_n)$. Likewise, we use F/n and $F(s_1, \ldots, s_n) : s_{n+1}$ to denote that function symbol F has arity n, respectively sorts $(s_1, \ldots, s_{n+1})$.
Intuitively, sorts are used to denote classes of "objects", predicate symbols to denote relations between these objects and function symbols to denote functions from (tuples of) objects to objects.

Example 2.2. The following is a suitable vocabulary Σ to describe unsolved battleship puzzles.

- $\Sigma_{\text{sort}} = \{\mathit{Row}, \mathit{Col}, \mathit{Number}, \mathit{Ship}, \mathit{Length}\}$. We will use these sorts to denote respectively the rows and columns of the grid, their associated numbers, the available ships and the possible lengths of ships.

- $\Sigma_{\text{pred}} = \{\mathit{ShipHint}(\mathit{Row}, \mathit{Col}), \mathit{WaterHint}(\mathit{Row}, \mathit{Col})\}$. The intension is to use these predicates to describe the given partial solution. That is, *ShipHint* denotes the squares (r, c) in the grid that contain part of a ship, while *WaterHint* denotes the squares that contain water.

[3] The requirement that Σ' has exactly the same sorts as Σ is introduced for a technical reason only. It ensures that in a model expansion problem (Definition 4.15), the input structure I_σ fixes the whole domain of the solution.

- $\Sigma_{\text{func}} = \{\mathit{RowNumber}(\mathit{Row}) : \mathit{Number}, \mathit{ColNumber}(\mathit{Col}) : \mathit{Number}, \mathit{ShipLength}(\mathit{Ship}) : \mathit{Length}\}$. These functions will be used to denote the mapping of, respectively, a row to its associated number, a column to its associated number and a ship to its length.

A Σ-*structure* associates appropriate values to the sorts, predicate and function symbols of vocabulary Σ. That is, a Σ-structure I consists of:

- A non-empty set s^I for each sort $s \in \Sigma_{\text{sort}}$. This set is called the *domain of s in I* and its elements are called *domain elements*.
- A relation $P^I \subseteq (s_1^I \times \cdots \times s_n^I)$ for each predicate symbol $P(s_1, \ldots, s_n) \in \Sigma_{\text{pred}}$.
- A function $F^I : (s_1^I \times \cdots \times s_n^I) \to s_{n+1}^I$ for each of the function symbols $F(s_1, \ldots, s_n) : s_{n+1} \in \Sigma_{\text{func}}$.

We call the value assigned by I to a symbol also the *interpretation* of that symbol in I. If $(s_1, \ldots, s_n)$ is a tuple of sorts, we denote by $(s_1, \ldots, s_n)^I$ the set $(s_1^I \times \cdots \times s_n^I)$. We call I a *finite* Σ-structure if the domain of s in I is finite for every sort $s \in \Sigma_{\text{sort}}$.

Example 2.3. Let Σ be the vocabulary defined in Example 2.2. Then the following is a Σ-structure I that describes the battleship puzzle shown in Figure 2.1. The sets associated to the sorts of Σ are given by

$$\begin{aligned}
\mathit{Row}^I &= \{1,2,3,4,5,6,7,8,9,10\} \\
\mathit{Col}^I &= \{1,2,3,4,5,6,7,8,9,10\} \\
\mathit{Number}^I &= \{0,1,2,3,4,5,6\} \\
\mathit{Ship}^I &= \{\mathit{Battleship}, \mathit{Cruiser1}, \mathit{Cruiser2}, \mathit{Destroyer1}, \ldots\} \\
\mathit{Length}^I &= \{1,2,3,4\}.
\end{aligned}$$

The two hints in the puzzle are given by the relations associated to the predicates symbols:

$$\begin{aligned}
\mathit{ShipHint}^I &= \{(2,3),(3,3)\} \\
\mathit{WaterHint}^I &= \{(1,3),(9,5)\}
\end{aligned}$$

Note that a structure for the chosen vocabulary Σ cannot directly describe that square $(2,3)$ contains "the first part of a ship of length at least two which is heading north". However, given the rules of battleship puzzle, this is equivalent to stating that $(1,3)$ contains no ship and both $(2,3)$ and $(3,3)$ contain a ship. We briefly come back to this issue later on in the text. Finally, I associates to each function symbol the intended functions.

$$\begin{aligned}
\mathit{RowNumber}^I &= \{1 \mapsto 2, 2 \mapsto 1, 3 \mapsto 5, \ldots\} \\
\mathit{ColNumber}^I &= \{1 \mapsto 0, 2 \mapsto 1, 3 \mapsto 2, \ldots\} \\
\mathit{ShipLength}^I &= \{\mathit{Battleship} \mapsto 4, \mathit{Cruiser1} \mapsto 3, \ldots\}
\end{aligned}$$

Observe that a predicate P with arity 0 can only have two interpretations: each structure I either assigns the empty set $\emptyset$ or the set $\{()\}$, i.e., the set containing the empty tuple, to P.
If $\Sigma' \subseteq \Sigma$, then $I|_{\Sigma'}$ denotes the *restriction* of I to the symbols in Σ'. Vice versa, I is called an *expansion of* $I|_{\Sigma'}$ *to* Σ.

Syntax

For the rest of this section, let Σ be a vocabulary. A *term* over Σ is inductively defined by

- each variable $v \in \Sigma_{\text{var}}$ is a term of sort $\mathfrak{s}(v)$;
- if $t_1, \ldots, t_n$ are terms of sort respectively $s_1, \ldots, s_n$ and $F(s_1, \ldots, s_n) : s_{n+1} \in \Sigma_{\text{func}}$, then $F(t_1, \ldots, t_n)$ is a term of sort s_{n+1}.

A *formula* over a vocabulary Σ is inductively defined by

- if $t_1, \ldots, t_n$ are terms of sort $s_1, \ldots, s_n$ and $P(s_1, \ldots, s_n)$ is a predicate symbol, then $P(t_1, \ldots, t_n)$ is a formula;
- if t_1 and t_2 are two terms of the same sort, then $t_1 = t_2$ is a formula;
- if φ and ψ are formulas and x is a variable, then $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\forall x\ (\varphi))$ and $(\exists x\ (\varphi))$ are formulas.

The symbols '$\neg$' (negation), '$\wedge$' (conjunction) and '$\vee$' (disjunction) are called *connectives*, '$\forall$' and '$\exists$' are called *quantifiers*: '$\forall$' is the universal quantifier, '$\exists$' the existential quantifier. Informally, '$\neg$' stands for *not*, '$\wedge$' for *and*, '$\vee$' for *or*, '$\forall$' for *for all* and '$\exists$' for *there exists*. We omit parenthesis in formulas if no confusion is possible and we assume '$\neg$' binds stronger than '$\forall$' and '$\exists$', which in turn bind stronger than '$\wedge$' and '$\vee$'.

Example 2.4. The following are two formulas over the vocabulary introduced in Example 2.2:

$$\forall r \forall c\ (\neg(\mathit{WaterHint}(r, c) \wedge \mathit{ShipHint}(r, c))); \tag{2.1}$$

$$\exists s_1 \forall s_2\ (\neg(\mathit{ShipLength}(s_1) = \mathit{ShipLength}(s_2)) \vee s_1 = s_2), \tag{2.2}$$

where r, c, s_1 and s_2 are variables of sort, respectively, *Row*, *Col*, *Ship* and *Ship*. The first formula can be read as *For each row r and each column c it is not the case that the square (r, c) contains both water and a part of a ship.* The second formula says *There exists a ship s_1 such that any ship s_2 has either a different length than s_1 or is s_1 itself.* In other words, the second formula expresses that there is some ship with a unique length.

We use $(\varphi \Rightarrow \psi)$, $(\varphi \Leftrightarrow \psi)$ and $(t_1 \neq t_2)$ as shorthands for respectively $(\neg\varphi \vee \psi)$, $((\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi))$ and $\neg(t_1 = t_2)$. From now on, we use $\overline{d}$ to denote tuples of domain elements, $\overline{s}$ to denote tuples of sorts, $\overline{t}$ to denote tuples of terms and

$\overline{x}$, $\overline{y}$, ..., to denote both sets and tuples of variables. The length of a tuple $\overline{a}$, i.e., the number of elements in $\overline{a}$, is denoted by $|\overline{a}|$. If $\overline{x}$ is the tuple of variables $(x_1, \ldots, x_n)$, then $(\forall \overline{x}\ \varphi)$ and $(\exists \overline{x}\ \varphi)$ denote respectively the formulas $(\forall x_1 \cdots \forall x_n\ \varphi)$ and $(\exists x_1 \cdots \exists x_n\ \varphi)$. If $\bar{t}$ is the tuple of terms $(t_1, \ldots, t_n)$, then $\mathfrak{s}(\bar{t})$ denotes the tuple of sorts $(\mathfrak{s}(t_1), \ldots, \mathfrak{s}(t_n))$.
An *atomic formula*, also called an *atom*, is a formula of the form $P(\bar{t})$ or $t_1 = t_2$. A *literal* is an atom (positive literal) or the negation of an atom (negative literal).
A *subformula* is a part of a formula that is a formula itself. For example, if φ is the formula $(P(x) \wedge Q(y)) \vee R(x)$, then the subformulas of φ are $P(x)$, $Q(y)$, $R(x)$, $(P(x) \wedge Q(x))$ and φ itself. A formula ψ is a *strict subformula* of φ if it is a subformula of φ and $\psi \neq \varphi$. A *direct subformula of* φ is a strict subformula of φ that is not a strict subformula of a strict subformula of φ.
In a formula of the form $(\forall x\ \varphi)$ or $(\exists x\ \varphi)$, the subformula φ is called the *scope* of '$\forall x$', respectively '$\exists x$'. Every occurrence of variable x in φ is said to be *bound* by '$\forall x$', respectively '$\exists x$'. If a formula φ contains an occurrence of a variable x that is not bound by any quantifier, x is a *free variable of* φ. We will often denote φ by $\varphi[\overline{x}]$ to indicate that $\overline{x}$ is the set of free variables of φ. A *sentence* is a formula that has no free variables.
If φ is a formula and term t occurs in φ, we denote by $\varphi[t/t']$ the formula obtained by replacing all occurrences of term t in φ by term t'. When we use this notation, we assume that all of the variables in t' are free in $\varphi[t/t']$. We extend this notation to tuples of terms of the same length.

Semantics

Before we can present the semantics of FO, we still need to introduce the following concept. A *variable assignment for a Σ-structure I* is a function θ mapping each variable $x \in \Sigma_{\text{var}}$ to a domain element of sort $\mathfrak{s}(x)$. If x is a variable and $d \in \mathfrak{s}(x)^I$, then we denote by $\theta[x/d]$ the variable assignment that assigns d to x and corresponds to θ on all other variables. This notation is extended to tuples of variables and domain elements of the same length. If $\overline{x}$ is the tuple of variables $(x_1, \ldots, x_n)$, then we denote the tuple of domain elements $(\theta(x_1), \ldots, \theta(x_n))$ also by $\theta(\overline{x})$.
The semantics of FO is defined by the satisfaction relation $\models$. This relation states which formulas over Σ are true in a given Σ-structure I under a variable assignment θ. The *value of a term t in a Σ-structure I under θ* is denoted by $I\theta(t)$ and defined by

- $I\theta(t) = \theta(t)$ if t is a variable;
- $I\theta(F(t_1, \ldots, t_n)) = F^I(I\theta(t_1), \ldots, I\theta(t_n))$.

If $\bar{t}$ is the tuple of terms $(t_1, \ldots, t_n)$, we denote by $I\theta(\bar{t})$ the tuple of domain elements $(I\theta(t_1), \ldots, I\theta(t_n))$. The satisfaction relation is defined by structural induction:

- $I\theta \models P(\bar{t})$ if $(I\theta(\bar{t})) \in P^I$;

- $I\theta \models (t_1 = t_2)$ if $I\theta(t_1) = I\theta(t_2)$;
- $I\theta \models (\neg\varphi)$ if $I\theta \not\models \varphi$;
- $I\theta \models (\varphi \wedge \psi)$ if $I\theta \models \varphi$ and $I\theta \models \psi$;
- $I\theta \models (\varphi \vee \psi)$ if $I\theta \models \varphi$ or $I\theta \models \psi$;
- $I\theta \models (\forall x\ \varphi)$ if $I\theta[x/d] \models \varphi$ for every $d \in \mathfrak{s}(x)^I$;
- $I\theta \models (\exists x\ \varphi)$ if there exists a $d \in \mathfrak{s}(x)^I$ such that $I\theta[x/d] \models \varphi$.

A formula φ is *satisfiable* if there exists a structure I and variable assignment θ such that $I\theta \models \varphi$. As mentioned, $I\theta \models \varphi$ intuitively means that statement φ is true in I, given that the variables of φ are interpreted according to θ. Observe that if $\overline{x}$ are the free variables of φ and $y \notin \overline{x}$, then it does not depend on $\theta(y)$ whether $I\theta \models \varphi$ or not. That is, $I\theta_1[\overline{x}/\overline{d}] \models \varphi$ iff $I\theta_2[\overline{x}/\overline{d}] \models \varphi$ for any two variable assignments θ_1 and θ_2. In this case, we often omit θ and simply write $I[\overline{x}/\overline{d}] \models \varphi$. In particular, if φ is a sentence we write $I \models \varphi$ and say that I satisfies φ.

Example 2.5. Let I be the structure defined in Example 2.3. Then both sentences (2.1) and (2.2) are satisfied in I. Indeed, for any variable assignment θ, $I\theta \models \mathit{WaterHint}(r, c)$ iff $\theta(r) = 9$ and $\theta(c) = 5$. Likewise, $I\theta \models \mathit{ShipHint}(r, c)$ iff $\theta(r) = 2$ and $\theta(c) = 3$. Hence $I\theta \not\models (\mathit{WaterHint}(r, c) \wedge \mathit{ShipHint}(r, c))$ for any θ. It follows that $I \models (2.1)$. Similarly, it can be checked that $I \models (2.2)$.

An important property of first-order logic as the basis for a KBS language is that the informal reading of FO sentences and the formal semantics correspond. It ensures we can rely on the informal reading when writing FO sentences and do not have to worry about the formal semantics. For example, we indicated that sentence (2.2) informally expresses that there is a ship with a unique length. Clearly, this statement is true in the puzzle depicted in Figure 2.1 since it contains a unique ship of length four. This corresponds to the formal semantics: the structure I from Example 2.3 describes the situation in Figure 2.1 and $I \models (2.2)$.

We denote by $\top$, respectively $\bot$, a sentence that is satisfied, respectively not satisfied, in every structure. That is, $\top$ and $\bot$ are shorthands for, e.g., $(\forall x\ x = x)$, respectively $(\exists x\ x \neq x)$. Equivalently, $\top$ and $\bot$ can be seen as predicates with arity 0 that have a fixed interpretation in every structure I: $\top^I$ is the set containing the empty tuple, $\bot^I$ is the empty set.

A *theory* T is a finite set of sentences. A structure satisfies T, denoted $I \models T$, if I satisfies each sentence of T. A structure that satisfies a theory T is also called a *model* of T. If T_1 and T_2 are two theories, we denote by $T_1 \models T_2$ that every model of T_1 is also a model of T_2.

A *set expression* is an expression of the form $\{\overline{x} \mid \varphi[\overline{y}]\}$ where φ is a formula and $\overline{y} \subseteq \overline{x}$. The interpretation $\{\overline{x} \mid \varphi[\overline{y}]\}^I$ of set expression $\{\overline{x} \mid \varphi[\overline{y}]\}$ in structure I is the set $\{\overline{d} \mid I[\overline{x}/\overline{d}] \models \varphi\}$. We will sometimes call set expressions *queries*. Tuples in the set $\{\overline{x} \mid \varphi[\overline{y}]\}^I$ are then called *answers* to the query $\{\overline{x} \mid \varphi\}$ in I.

Complexity of the satisfaction relation

In general, it is undecidable whether a given structure I satisfies a given theory T. Similarly, the question whether a given theory T implies a given sentence φ, i.e., the question whether $T \models \varphi$, is undecidable. If structure I has a finite domain however, it is decidable whether I satisfies a given theory T.
In the context of finite structures, there are three different classes of decision problems related to the satisfaction relation $\models$ of a logic $\mathcal{L}$ (Vardi, 1982). The *combined complexity of* $\mathcal{L}$ is the problem of deciding for a given finite structure I and $\mathcal{L}$-sentence φ whether $I \models \varphi$. The *data-complexity of* $\mathcal{L}$ is a class of decision problems $\mathcal{Q}_\varphi$, one for each $\mathcal{L}$-sentence φ. Each of these problems $\mathcal{Q}_\varphi$ is the problem of deciding for a given finite structure I whether $I \models \varphi$. That is, for data-complexity the sentence φ is considered to be fixed and the complexity of deciding $I \models \varphi$ is considered as a function of I. Vice versa, the *expression complexity of* L is the class of decision problems $\mathcal{Q}_I$, one for each finite structure I, where $\mathcal{Q}_I$ is the problem of deciding for a given sentence φ whether $I \models \varphi$.
The following theorem states the complexity of deciding these problems if $\mathcal{L}$ is the logic FO.

Theorem 2.1. *For FO, the following hold:*

- *The combined complexity of FO is* **PSPACE***-complete.*
- *The data complexity of FO is in* **L***, i.e., for any fixed sentence φ and given finite structure I, it can be decided in logarithmic space in the size of I if $I \models \varphi$.*
- *The expression complexity of FO is* **PSPACE***-complete.*

Since $\mathbf{L} \subseteq \mathbf{P}$*, it follows that the data-complexity of FO is in* $\mathbf{P}$*.*

Proof. See, e.g., the proof of Theorem 2.4.3 in the book of Grädel et al. (2007). ■

2.1.2 One-sorted and zero-sorted logic

While we will use many-sorted FO in most of the examples in this thesis, we will often use *one-sorted FO* to facilitate the presentation in the theoretical parts. In one-sorted FO, each vocabulary has only one sort. Hence in one-sorted vocabulary Σ, the sort of a predicate or function symbol is completely determined by its arity. We introduce one-sorted vocabularies by expressions of the form $\Sigma = \langle V_p, V_f \rangle$, stating that Σ_{pred} is the set V_p and Σ_{func} the set V_f. The unique sort is left implicit. The domain assigned by a Σ-structure I to the unique sort in Σ is called *the domain of* I.[4] Oberschelp (1962) showed that many-sorted FO can be reduced to one-sorted FO by introducing a predicate of arity one for each sort of a many-sorted vocabulary. We recall his construction in Section 3.2.

[4] If in a many-sorted context, we refer to *the* domain of a Σ-structure I, we mean $I|_{\Sigma_{\text{sort}}}$, i.e., the set of the domains of all sorts in Σ.

Another important fragment of FO is *propositional logic* (PC). In a propositional vocabulary, the set of sorts is empty, all predicate symbols have arity 0 and there are neither function symbols nor variables. A *propositional theory* is a theory over a propositional vocabulary. A propositional *clause* is a disjunction $L_1 \vee \ldots \vee L_n$ where all L_i, $1 \leq i \leq n$, are propositional literals. A propositional theory is in *conjunctive normal form* (CNF) if all its sentences are clauses.
The *boolean satisfiability problem* (SAT) is the problem of deciding for a propositional theory whether it has a model. This problem is **NP**-complete. Hence each decision problem in **NP** can be reduced in polynomial time to a SAT problem. Contemporary SAT solvers (Mitchell, 2005) exhibit impressive performance. As such, many problems in **NP** can be solved efficiently by reducing them to SAT. For instance, this is done in the areas of model generation (Claessen and Sörensson, 2003; McCune, 2003), planning (Kautz and Selman, 1996) and relational data mining (Krogel et al., 2003). Most modern SAT solvers expect a CNF theory as input, instead of a general PC theory. When the input is a satisfiable theory T, they return a model of T as a witness to their answer.

2.1.3 Equivalence, rewriting and normal forms

In this section we introduce different notions of "equivalence" and several rules to rewrite formulas into equivalent formulas. Rewriting is applied to transform formulas in some desired normal form which may, e.g., facilitate the presentation of theorems, proofs and algorithms. We introduce one such normal form in this section.

Logical equivalence

A formula φ that is satisfied in any structure and variable assignment is called *valid*. This is denoted by $\models \varphi$. Two formulas φ_1 and φ_2 are *logically equivalent* if $\varphi_1 \Leftrightarrow \varphi_2$ is a valid formula.[5] Hence φ_1 and φ_2 are logically equivalent if for any structure I and variable assignment θ, $I\theta \models \varphi_1$ iff $I\theta \models \varphi_2$. The following are important pairs of logically equivalent formulas.

1. Moving quantifiers

$$\forall x \forall y\ \varphi \quad \Leftrightarrow \quad \forall y \forall x\ \varphi \tag{2.3}$$
$$\exists x \exists y\ \varphi \quad \Leftrightarrow \quad \exists y \exists x\ \varphi \tag{2.4}$$
$$\forall x\ (\varphi \wedge \psi) \quad \Leftrightarrow \quad (\forall x\ \varphi) \wedge (\forall x\ \psi) \tag{2.5}$$
$$\exists x\ (\varphi \vee \psi) \quad \Leftrightarrow \quad (\exists x\ \varphi) \vee (\exists x\ \psi) \tag{2.6}$$
$$\forall x\ (\varphi \vee \psi) \quad \Leftrightarrow \quad \varphi \vee (\forall x\ \psi) \quad \text{if } x \text{ does not occur free in } \varphi \tag{2.7}$$
$$\exists x\ (\varphi \wedge \psi) \quad \Leftrightarrow \quad \varphi \wedge (\exists x\ \psi) \quad \text{if } x \text{ does not occur free in } \varphi \tag{2.8}$$

[5] In the rest of this thesis we will introduce several notions of "equivalence". When we say that two formulas are equivalent, we always mean *logically* equivalent

2. Moving negations

$$\neg(\varphi \wedge \psi) \quad \Leftrightarrow \quad (\neg\varphi) \vee (\neg\psi) \tag{2.9}$$
$$\neg(\varphi \vee \psi) \quad \Leftrightarrow \quad (\neg\varphi) \wedge (\neg\psi) \tag{2.10}$$
$$\neg(\forall x\ \varphi) \quad \Leftrightarrow \quad \exists x\ (\neg\varphi) \tag{2.11}$$
$$\neg(\exists x\ \varphi) \quad \Leftrightarrow \quad \forall x\ (\neg\varphi) \tag{2.12}$$

3. Flattening terms

$$\varphi \quad \Leftrightarrow \quad \exists x\ (x = t \wedge \varphi[t/x]) \tag{2.13}$$
$$\varphi \quad \Leftrightarrow \quad \forall x\ (x = t \Rightarrow \varphi[t/x]) \tag{2.14}$$

where x is a "fresh" variable, i.e., x does not occur in φ.

These (and other) equivalences can be used to rewrite a formula φ_1 into a logically equivalent formula φ_2. For example, if φ_1 is the formula $(\forall x \forall y\ \varphi)$, then it follows from (2.3) that rewriting it to the formula $(\forall y \forall x\ \varphi)$ preserves equivalence. Rewriting can be applied to transform a formula in an equivalent one that has a desired format. We will define several such formats throughout this thesis. One format is called *term normal form*.

Definition 2.1. A formula φ is in *term normal form* (TNF) if all negations in φ occur directly in front of atoms, and all atomic subformulas of φ are of the form $P(x_1, \ldots, x_n)$, $F(x_1, \ldots, x_n) = y$ or $x = y$, where $x, y, x_1, \ldots, x_n$ are variables, P is a predicate symbol and F a function symbol.

Every formula can be rewritten to a logically equivalent formula in TNF by applying the equivalences above: first bring all atomic subformulas in TNF by applying (2.13) or (2.14), then move all negations inside using (2.9)–(2.12).

Σ-equivalence

Two formulas φ_1 and φ_2 are *equisatisfiable* if φ_1 is satisfiable iff φ_2 is satisfiable. Clearly, if φ_1 and φ_2 are logically equivalent, then they are also equisatisfiable. We now introduce a form of equivalence that lies in between logical equivalence and equisatisfiability.

Definition 2.2. Let Σ_1 and Σ_2 be two vocabularies that share a common subvocabulary Σ and let φ_1 and φ_2 be sentences over, respectively, Σ_1 and Σ_2. Then φ_1 and φ_2 are Σ-*equivalent* if for any Σ-structure I there exists an expansion M_1 of I to Σ_1 such that $M_1 \models \varphi_1$ iff there exists an expansion M_2 of I to Σ_2 such that $M_2 \models \varphi_2$.

Observe that if $\Sigma_1 = \Sigma_2$ in the definition above, Σ-equivalence corresponds to logical equivalence of sentences. Vice versa, two logically equivalent sentences are also Σ-equivalent for any vocabulary Σ. On the other hand, if Σ is the empty vocabulary then φ_1 and φ_2 are Σ-equivalent iff they are equisatisfiable. The definition of Σ-equivalence is extended to theories in the obvious way.
The following proposition presents a method to rewrite sentences to Σ-equivalent sentences.

Proposition 2.2. *Let φ be a sentence over a vocabulary Σ and let $\psi[\overline{x}]$ be a subformula of φ. Let $P(\mathfrak{s}(\overline{x}))$ be a predicate not occurring in Σ and denote by φ' the result of replacing $\psi[\overline{x}]$ by $P(\overline{x})$ in φ. Then $\varphi' \wedge \forall\overline{x}(P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$ is Σ-equivalent to φ.*

Proof. Denote the vocabulary $\Sigma \cup \{P\}$ by Σ' and let I be a Σ-structure. Any expansion of I to Σ' that satisfies the sentence $\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$ necessarily assigns $\{\overline{x} \mid \psi[\overline{x}]\}^I$ to P. Hence, such an expansion satisfies φ' iff $I \models \varphi$. ■

The process applied in Proposition 2.2, i.e., replacing a complex subformula ψ by a new predicate P and simultaneously providing a "definition" $(\forall\overline{x}\ P(\overline{x}) \Leftrightarrow \psi)$ for this new predicate is called *predicate introduction*. Predicate introduction can be used to rewrite an arbitrary propositional theory to a CNF theory in linear time.[6] This method was introduced by Tseitin (1968).

Algorithm 2.1 (Tseitin transformation)**.** Let T be a propositional theory over propositional vocabulary Σ. Then the following algorithm rewrites T to a Σ-equivalent CNF theory.

1. Move all negations inside using (2.9) and (2.10) until they are directly in front of atoms.
2. Replace every sentence of the form $\varphi_1 \wedge \ldots \wedge \varphi_n$ by the sentences $\varphi_1, \ldots, \varphi_n$. Omit as many brackets as possible. Now all sentences in T are of the form $\psi_1 \vee \ldots \vee \psi_m$ where each of the ψ_i, $1 \leq i \leq m$ is either a conjunction or a literal.
3. While T is not in CNF, repeat the following steps:
   (a) Choose a sentence φ of the form $\psi_1 \vee \ldots \vee \psi_m$ from T that is not a clause and choose one of its direct subformulas ψ_i, $1 \leq i \leq m$, that is not a literal. This subformula is of the form $\chi_1 \wedge \ldots \wedge \chi_n$.
   (b) Replace φ by $\psi_1 \vee \ldots \vee \psi_{i-1} \vee P \vee \psi_{i+1} \vee \ldots \vee \psi_m$, where P is a new predicate symbol.
   (c) Add the sentences $(\neg P \vee \chi_1), \ldots, (\neg P \vee \chi_n)$ and $(\neg\chi_1 \vee \ldots \vee \neg\chi_n \vee P)$ to T, and move in these sentence all negations inside until they are directly in front of atoms.
4. Return T.

Proposition 2.3. *On input theory T over propositional vocabulary Σ, Algorithm 2.1 returns a theory T' in CNF that is Σ-equivalent to T.*

Proof. Observe that the combination of steps (3b) and (3c) implements predicate introduction. Indeed, the combination of all sentences added in step (3c) is logically equivalent to the sentence $P \Leftrightarrow (\chi_1 \wedge \ldots \wedge \chi_n)$. The proof of Proposition 2.3 now easily follows from Proposition 2.2. ■

[6] On the other hand, the well-known method of converting a propositional theory to CNF by distributing disjunction over conjunction takes exponential time.

2.1.4 Function graphs

Several times in this thesis, we will facilitate the presentation by assuming that vocabularies do not contain function symbols. Often this assumption can be made without loss of generality, since in any theory T function symbols can be replaced by predicate symbols, provided that some extra sentences are added to T.

Formally, let $\mathscr{G}(\Sigma)$ be the vocabulary obtained from Σ by replacing each function symbol $F(\overline{s}) : s'$ by a new predicate symbol $\mathscr{G}_F(\overline{s}, s')$. That is, $\mathscr{G}(\Sigma)_{\text{sort}} = \Sigma_{\text{sort}}$, $\mathscr{G}(\Sigma)_{\text{pred}} = \Sigma_{\text{pred}} \cup \{\mathscr{G}_F(\overline{s}, s') \mid F(\overline{s}) : s' \in \Sigma_{\text{func}}\}$, $\mathscr{G}(\Sigma)_{\text{func}} = \emptyset$ and $\mathscr{G}(\Sigma)_{\text{var}} = \Sigma_{\text{var}}$. The predicates $\mathscr{G}_F$ are used to denote the *graph* of the function denoted by F. If I is a Σ-structure, we denote by $\mathscr{G}(I)$ the $\mathscr{G}(\Sigma)$-structure that assigns $\mathscr{G}_F^{\mathscr{G}(I)} = \{(\overline{d}, d') \mid F^I(\overline{d}) = d'\}$ for every $F \in \Sigma_{\text{func}}$ and $P^{\mathscr{G}(I)} = P^I$ for every $P \in \Sigma_{\text{pred}}$.

For a $\mathscr{G}(\Sigma)$-structure J, there does not necessarily exists a Σ-structure I such that $\mathscr{G}(I) = J$. If there exists such a structure I, it is unique. In this case, we say that J is *function consistent*, and we denote I by $\mathscr{G}^{-1}(J)$. A structure J is function consistent iff for any predicate $\mathscr{G}_F(\overline{s}, s')$ in its vocabulary and every $\overline{d} \in \overline{s}^J$, there is exactly one $d' \in (s')^J$ such that $(\overline{d}, d') \in \mathscr{G}_F^J$. Equivalently, J is function consistent if for every predicate $\mathscr{G}_F$ in its vocabulary, J satisfies the sentences

$$\forall \overline{x} \exists y \; \mathscr{G}_F(\overline{x}, y); \tag{2.15}$$

$$\forall \overline{x} \forall y_1 \forall y_2 \; (\mathscr{G}_F(\overline{x}, y_1) \wedge \mathscr{G}_F(\overline{x}, y_2) \Rightarrow y_1 = y_2). \tag{2.16}$$

Proposition 2.4. *For every theory T over Σ there exists a theory T' over $\mathscr{G}(\Sigma)$ such that any model of T' is function consistent and $I \models T$ iff $\mathscr{G}(I) \models T'$.*

Proof. Let T' be the theory obtained by first converting T to TNF, then replacing each atomic subformula $F(\overline{x}) = y$ by $\mathscr{G}_F(\overline{x}, y)$ and finally adding the sentences (2.15) and (2.16) for each function symbol F. Clearly, every model of T' is function consistent. Since $\{(\overline{x}, y) \mid F(\overline{x}) = y\}^I = \{(\overline{x}, y) \mid \mathscr{G}_F(\overline{x}, y)\}^{\mathscr{G}(I)}$ for any Σ-structure I, it follows by induction that $I \models T$ iff $\mathscr{G}(I) \models T'$. ■

Proposition 2.4 allows us to assume without loss of generality that a theory does not contain function symbols. Therefore we make no distinction between I and $\mathscr{G}(I)$ for the rest of this thesis, unless stated otherwise. Also, if formula φ' is obtained by replacing in a formula φ an atom $F(\overline{t}) = t'$ by $\mathscr{G}_F(\overline{t}, t')$, φ and φ' are considered equal. If φ is a TNF formula, we denote by $\mathscr{G}(\varphi)$ the result of replacing all atomic subformulas of the form $F(\overline{x}) = y$ in φ by $\mathscr{G}_F(\overline{x}, y)$.

2.2 Three- and four-valued structures

In this section we present more general structures than the ones defined in Section 2.1.1. In these more general structures it is possible to express partial and inconsistent information.

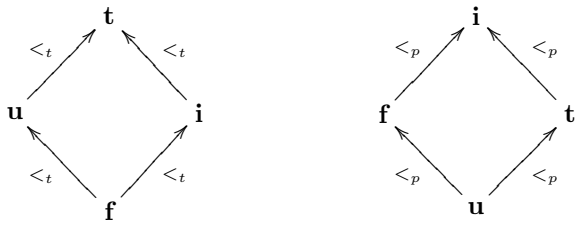


Figure 2.3: The truth and precision order

2.2.1 Four-valued structures

Denote the truth values *true*, *false*, *unknown* and *inconsistent* by respectively **t**, **f**, **u** and **i**. For a truth value **v**, the *inverse value* $\mathbf{v}^{-1}$ is defined by $\mathbf{t}^{-1} = \mathbf{f}$, $\mathbf{f}^{-1} = \mathbf{t}$, $\mathbf{u}^{-1} = \mathbf{u}$ and $\mathbf{i}^{-1} = \mathbf{i}$. The *truth order* $<_t$ and the *precision order* $<_p$, also called *knowledge order*, are defined in Figure 2.3. The reflexive closure of these orders is denoted by $\leq_t$, respectively $\leq_p$.
We now introduce *four-valued structures*. In such structures it is possible to express that it is unknown whether some tuple belongs to a relation, or whether the situation described by the structure is inconsistent. Formally a four-valued Σ-structure $\tilde{I}$ consists of

- a domain $s^{\tilde{I}}$ for every $s \in \Sigma_{\text{sort}}$;
- a function $P^{\tilde{I}} : \overline{s}^{\tilde{I}} \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}, \mathbf{i}\}$ for every $P(\overline{s}) \in \Sigma_{\text{pred}}$;
- a function $F^{\tilde{I}} : \overline{s}^{\tilde{I}} \to \mathscr{P}((s')^{\tilde{I}})$ for every $F(\overline{s}) : s' \in \Sigma_{\text{func}}$.

Here, $\mathscr{P}((s')^{\tilde{I}})$ denotes the power set of the set $(s')^{\tilde{I}}$. Intuitively, $P^{\tilde{I}}(\overline{d}) = \mathbf{u}$ means that it is unknown whether tuple $\overline{d}$ belongs to the relation denoted by P. Similarly, $P^{\tilde{I}}(\overline{d}) = \mathbf{i}$ means that it is an inconsistency that $\overline{d}$ belongs to the relation. If $d' \in F^{\tilde{I}}(\overline{d})$, then d' is a possible image of $\overline{d}$ under the function denoted by F. It is only certain that d' is the image of $\overline{d}$ under F if $F^{\tilde{I}}(\overline{d})$ is the singleton $\{d'\}$. If $F^{\tilde{I}}(\overline{d}) = \emptyset$, $\overline{d}$ has no possible image, which corresponds to an inconsistency.
A four-valued structure $\tilde{I}$ is called *three-valued* if it contains no inconsistencies. That is, $P^{\tilde{I}}(\overline{d}) \neq \mathbf{i}$ and $F^{\tilde{I}}(\overline{d}) \neq \emptyset$ for any predicate symbol P, function symbol F and tuple of domain elements $\overline{d}$. A three-valued structure $\tilde{I}$ is *two-valued* if $P^{\tilde{I}}(\overline{d}) \neq \mathbf{u}$ and $F(\overline{d})$ is a singleton for every predicate symbol P, function symbol F and tuple of domain elements $\overline{d}$. A two-valued structure $\tilde{I}$ can be seen as the structure I defined by $\overline{d} \in P^I$ iff $P^{\tilde{I}}(\overline{d}) = \mathbf{t}$ and $F^I(\overline{d}) = d'$ iff $F^{\tilde{I}}(\overline{d}) = \{d'\}$ for every P, F and $\overline{d}$. In the rest of this thesis we make no distinction between two-valued structures as defined in this section and structures as defined in Section 2.1.1. We call a structure *strictly three-valued* if it is three-valued but not two-valued. Likewise, a structure is *strictly four-valued* if it is four-valued but not three-valued.

A four-valued Σ-structure $\tilde{I}$ is two-valued (three-valued) on a predicate symbol P if $\tilde{I}|_{\langle \Sigma_{\text{sort}}, \{P\}, \emptyset \rangle}$ is two-valued (three-valued). Similarly, $\tilde{I}$ is two-valued (three-valued) on a function symbol F if $\tilde{I}|_{\langle \Sigma_{\text{sort}}, \emptyset, \{F\} \rangle}$ is two-valued (three-valued).
In the rest of this thesis, three- or four-valued structures are always denoted by an uppercase letter and tilde. The tilde is only omitted in case of two-valued structures. As such, a phrase like "Let I be a structure" implicitly means "Let I be a *two-valued* structure", while "Let $\tilde{I}$ be a structure" means "Let $\tilde{I}$ be a four-valued (hence possibly three- or two-valued) structure".
The precision order pointwise extends to structures: if $\tilde{I}$ and $\tilde{J}$ are two Σ-structures, $\tilde{I} \leq_p \tilde{J}$ if for every sort s, predicate symbol P, function symbol F and tuple of domain elements $\overline{d}$, $s^{\tilde{I}} = s^{\tilde{J}}$, $P^{\tilde{I}}(\overline{d}) \leq_p P^{\tilde{J}}(\overline{d})$ and $F^{\tilde{I}}(\overline{d}) \supseteq F^{\tilde{J}}(\overline{d})$. For a fixed domain D, the most precise structure with domain D is denoted by $\top_D^{\leq_p}$ and assigns $P^{\top_D^{\leq_p}}(\overline{d}) = \mathbf{i}$ and $F^{\top_D^{\leq_p}}(\overline{d}) = \emptyset$ to every predicate symbol P, function symbol F and tuple of domain elements $\overline{d}$. Vice versa, the least precise structure $\bot_D^{\leq_p}$ assigns $P^{\bot_D^{\leq_p}} = \mathbf{u}$ and $F^{\bot_D^{\leq_p}}(\overline{d}) = D$ to every P, F and $\overline{d}$. If D is clear from the context, we write $\top^{\leq_p}$ and $\bot^{\leq_p}$ instead of $\top_D^{\leq_p}$ and $\bot_D^{\leq_p}$. If a two-valued structure I is more precise than three-valued structure $\tilde{I}$, we say that $\tilde{I}$ *approximates* I.

Example 2.6. Instead of using predicates *WaterHint* and *ShipHint* and a two-valued structure to describe an unsolved battleship puzzle, we could as well introduce the predicate $\mathit{ContainsShip}(\mathit{Row}, \mathit{Col})$ with the intended meaning that *ContainsShip* denotes the squares that contain part of a ship. The situation in Figure 2.1 is then represented by a three-valued structure $\tilde{I}$ that assigns

$$\mathit{ContainsShip}(r,c) = \begin{cases} \mathbf{t} & \text{if } (r,c) \in \{(2,3),(3,3)\} \\ \mathbf{f} & \text{if } (r,c) \in \{(1,3),(9,5)\} \\ \mathbf{u} & \text{for all other squares} \end{cases}$$

The structure $\tilde{I}$ expresses that $(2,3)$ and $(3,3)$ certainly contains part of a ship, $(1,3)$ and $(9,5)$ certainly contains no ship, and for the rest of the squares, it is still unknown whether they contain part of a ship. A solution to the battleship puzzle is then described by a two-valued structure I that is more precise than $\tilde{I}$ (and satisfies the rules of the puzzle). In other words, $\tilde{I}$ represents a partial solution.

The *domain size* $|\tilde{I}|$ of a structure $\tilde{I}$ is defined as the sum of the cardinalities of the domains assigned by $\tilde{I}$. The *full size* $\|\tilde{I}\|$ is the sum of the cardinalities of the relations assigned by $\tilde{I}$.
A *domain atom* over a structure $\tilde{I}$ is an expression of the form $P(\overline{d})$ where P is a predicate and $\overline{d} \in \mathfrak{s}(P)^{\tilde{I}}$. For a truth value $\mathbf{v}$ and domain atom $P(\overline{d})$, we denote by $\tilde{I}[P(\overline{d})/\mathbf{v}]$ the structure that assigns $P^{\tilde{I}}(\overline{d}) = \mathbf{v}$ and corresponds to $\tilde{I}$ on the rest of the vocabulary. A *domain literal* is a domain atom $P(\overline{d})$ or the negation $\neg P(\overline{d})$ of a domain atom. We denote by $\mathrm{atom}(L)$ the domain atom $P(\overline{d})$ if $L = P(\overline{d})$ or $L = \neg P(\overline{d})$. By $\tilde{I}[\neg P(\overline{d})/\mathbf{v}]$ we denote the structure $\tilde{I}[P(\overline{d})/\mathbf{v}^{-1}]$.

This notation is extended to sets of domain literals: if U is the set $\{L_1, \ldots, L_n\}$ of domain literals, $\tilde{I}[U/\mathbf{v}]$ denotes the structure $\tilde{I}[L_1/\mathbf{v}] \cdots [L_n/\mathbf{v}]$.
We now define the value of a TNF formula in a four-valued structure $\tilde{I}$.

Definition 2.3. The value $\tilde{I}\theta(\varphi)$ of a TNF formula φ in a four-valued structure $\tilde{I}$ under variable assignment θ is defined by structural induction:

- $\tilde{I}\theta(P(\overline{x})) = P^{\tilde{I}}(\theta(\overline{x}))$;
- $\tilde{I}\theta(F(\overline{x}) = y) = \begin{cases} \mathbf{i} & \text{if } F^{\tilde{I}}(\theta(\overline{x})) = \emptyset, \\ \mathbf{t} & \text{if } F^{\tilde{I}}(\theta(\overline{x})) = \{\theta(y)\}, \\ \mathbf{f} & \text{if } F^{\tilde{I}}(\theta(\overline{x})) \neq \emptyset \text{ and } \theta(y) \notin F^{\tilde{I}}(\theta(\overline{x})), \\ \mathbf{u} & \text{otherwise;} \end{cases}$
- $\tilde{I}\theta(\neg\varphi) = (\tilde{I}\theta(\varphi))^{-1}$;
- $\tilde{I}\theta(\varphi \wedge \psi) = \mathrm{glb}_{\leq_t}\{\tilde{I}\theta(\varphi), \tilde{I}\theta(\psi)\}$;
- $\tilde{I}\theta(\varphi \vee \psi) = \mathrm{lub}_{\leq_t}\{\tilde{I}\theta(\varphi), \tilde{I}\theta(\psi)\}$;
- $\tilde{I}\theta(\forall x\ \varphi) = \mathrm{glb}_{\leq_t}\{\tilde{I}\theta[x/d](\varphi) \mid d \in \mathfrak{s}(x)^{\tilde{I}}\}$;
- $\tilde{I}\theta(\exists x\ \varphi) = \mathrm{lub}_{\leq_t}\{\tilde{I}\theta[x/d](\varphi) \mid d \in \mathfrak{s}(x)^{\tilde{I}}\}$.

As in the two-valued case, the value of $\tilde{I}\theta(\varphi[\overline{x}])$ does not depend on the value of $\theta(y)$ for any variable $y \notin \overline{x}$. As such, we often omit θ and write $\tilde{I}[\overline{x}/\overline{d}](\varphi[\overline{x}])$ instead of $\tilde{I}\theta[\overline{x}/\overline{d}](\varphi[\overline{x}])$, and $\tilde{I}(\varphi)$ instead of $\tilde{I}\theta(\varphi)$ if φ is a sentence.
If $\tilde{I}$ is three-valued, then $\tilde{I}\theta(\varphi) \neq \mathbf{i}$ for every formula φ and variable assignment θ. One can check that the restriction of Definition 2.3 to three-valued structures corresponds to the standard Kleene semantics (Kleene, 1952). If $\tilde{I}$ is two-valued, then $\tilde{I}\theta(\varphi) \in \{\mathbf{t}, \mathbf{f}\}$. Also, if $\tilde{I}$ is two-valued, then $\tilde{I}\theta(\varphi) = \mathbf{t}$ iff $\tilde{I}\theta \models \varphi$. This shows that the definition of value of a formula and of the satisfaction relation coincide.
If φ is a formula and $\tilde{I}$ and $\tilde{J}$ are two structures such that $\tilde{I} \leq_p \tilde{J}$, then also $\tilde{I}\theta(\varphi) \leq_p \tilde{J}\theta(\varphi)$ for every θ. If φ is a formula that does not contain negations and $\tilde{I} \leq_t \tilde{J}$, then also $\tilde{I}\theta(\varphi) \leq_t \tilde{J}\theta(\varphi)$ for every θ. A similar property is expressed in the following lemma.

Lemma 2.5. *Let φ be a formula such that no function symbol occurs in φ and for every predicate symbol P, P occurs only positively (i.e., in the scope of an even number of negations) or only negatively (in the scope of an odd number of negations). Then for every three-valued structure $\tilde{I}$ and variable assignment θ such that $\tilde{I}\theta(\varphi) \geq_t \mathbf{u}$ there exists a two-valued structure M such that $M \geq_p \tilde{I}$ and $M\theta \models \varphi$.*

Proof idea. It suffices to take $M = \tilde{I}[U_1/\mathbf{t}][U_2/\mathbf{f}]$, where U_1 and U_2 are the set of domain atoms $P(\overline{d})$ such that $P^{\tilde{I}}(\overline{d}) = \mathbf{u}$ and P occurs positively, respectively negatively, in φ. ■

Defining the value of an arbitrary (non-TNF) formula in a four-valued structure is not without problems. Indeed, the value of a term t in a structure $\tilde{I}$ is not necessarily a domain element.[7] For example, if the constant C is inconsistent in $\tilde{I}$, i.e., $C^{\tilde{I}} = \{\}$, then the value of term C in $\tilde{I}$ cannot be a single domain element. It follows that defining, e.g., $P^{\tilde{I}}(\tilde{I}(t))$ is not straightforward. Also observe that different rewritings of a formula φ to TNF may have a different value in a structure $\tilde{I}$. For example, assume that φ is the sentence $P(C)$, $\tilde{I}$ has domain $\{d_1, d_2\}$, $P^{\tilde{I}}(d_1) = P^{\tilde{I}}(d_2) = \mathbf{t}$ and $C^{\tilde{I}} = \{d_1, d_2\}$. Then $\exists x\ (C = x \wedge P(x))$ and $\forall x\ (C \neq x \vee P(x))$ are two TNF sentences that are both equivalent to $P(C)$. Yet,

$$\begin{aligned}
&\tilde{I}(\exists x\ (C = x \wedge P(x))) \\
&= \mathrm{lub}_{\leq_t}\{\tilde{I}[x/d_1](C = x \wedge P(x)), \tilde{I}[x/d_2](C = x \wedge P(x))\} \\
&= \mathrm{lub}_{\leq_t}\left\{\begin{array}{l}\mathrm{glb}_{\leq_t}\{\tilde{I}[x/d_1](C = x), \tilde{I}[x/d_1](P(x))\}, \\ \mathrm{glb}_{\leq_t}\{\tilde{I}[x/d_2](C = x), \tilde{I}[x/d_2](P(x))\}\end{array}\right\} \\
&= \mathrm{lub}_{\leq_t}\{\mathrm{glb}_{\leq_t}\{\mathbf{u}, \mathbf{t}\}, \mathrm{glb}_{\leq_t}\{\mathbf{u}, \mathbf{t}\}\} \\
&= \mathrm{lub}_{\leq_t}\{\mathbf{u}, \mathbf{u}\} \\
&= \mathbf{u},
\end{aligned}$$

while

$$\begin{aligned}
&\tilde{I}(\forall x\ (C \neq x \vee P(x))) \\
&= \mathrm{glb}_{\leq_t}\{\tilde{I}[x/d_1](C \neq x \vee P(x)), \tilde{I}[x/d_2](C \neq x \vee P(x))\} \\
&= \mathrm{glb}_{\leq_t}\left\{\begin{array}{l}\mathrm{lub}_{\leq_t}\{\tilde{I}[x/d_1](C \neq x), \tilde{I}[x/d_1](P(x))\}, \\ \mathrm{lub}_{\leq_t}\{\tilde{I}[x/d_2](C \neq x), \tilde{I}[x/d_2](P(x))\}\end{array}\right\} \\
&= \mathrm{glb}_{\leq_t}\{\mathrm{lub}_{\leq_t}\{\mathbf{u}, \mathbf{t}\}, \mathrm{lub}_{\leq_t}\{\mathbf{u}, \mathbf{t}\}\} \\
&= \mathrm{glb}_{\leq_t}\{\mathbf{t}, \mathbf{t}\} \\
&= \mathbf{t}.
\end{aligned}$$

Note that the mentioned problems do not occur when evaluating formulas in structures that are two-valued on all function symbols, because in such a structure, the value of each term is indeed a single domain element. In some of the examples later on in the text, we use this fact and evaluate non-TNF sentences φ in four-valued structures that are two-valued on all function symbols in φ.

2.2.2 Function graphs

As in a two-valued context, also in a four-valued context it can be assumed that a vocabulary contains no function symbols. For a vocabulary Σ, define the function-free vocabulary $\mathscr{G}(\Sigma)$ as before. For a Σ-structure $\tilde{I}$ define the

[7] There is a similar problem when partial functions are allowed, see Section 3.1.1.

$\mathscr{G}(\Sigma)$-structure $\mathscr{G}(\tilde{I})$ by $P^{\tilde{I}} = P^{\mathscr{G}(\tilde{I})}$ for every predicate symbol P and

$$\mathscr{G}_F^{\mathscr{G}(\tilde{I})}(\overline{d}, d') = \begin{cases} \mathbf{i} & \text{if } F^{\tilde{I}}(\overline{d}) = \emptyset \\ \mathbf{t} & \text{if } F^{\tilde{I}}(\overline{d}) = \{d'\} \\ \mathbf{f} & \text{if } d' \notin F^{\tilde{I}}(\overline{d}) \text{ and } F^{\tilde{I}}(\overline{d}) \neq \emptyset \\ \mathbf{u} & \text{otherwise} \end{cases}$$

for every function symbol F. Note that $\mathscr{G}(\tilde{I})$ is three-valued (two-valued) iff $\tilde{I}$ is three-valued (two-valued). It follows directly from the definition above and Definition 2.3 that $\tilde{I}\theta(F(\overline{x}) = y) = \mathscr{G}(\tilde{I})\theta(\mathscr{G}_F(\overline{x}, y))$ for every function symbol F. By induction, it follows that for any TNF formula φ, $\mathscr{G}(\varphi)$ does not contain function symbols and $\tilde{I}\theta(\varphi) = \mathscr{G}(\tilde{I})\theta(\mathscr{G}(\varphi))$.
It is not necessarily the case that for a $\mathscr{G}(\Sigma)$ structure $\tilde{J}$, there exists a Σ-structure $\tilde{I}$ such that $\mathscr{G}(\tilde{I}) = \tilde{J}$. If such a structure $\tilde{I}$ exists, it is unique, and we denote it by $\mathscr{G}^{-1}(\tilde{J})$. We call $\tilde{J}$ *function consistent* in this case. The following proposition lists sufficient and necessary conditions for $\tilde{J}$ to be function consistent.

Proposition 2.6. *A $\mathscr{G}(\Sigma)$-structure $\tilde{J}$ is function consistent iff it satisfies all of the following conditions for every function symbol $F(\overline{s}) : s'$ and every $\overline{d} \in \overline{s}^{\tilde{J}}$:*

- *if there exists a $d' \in (s')^{\tilde{J}}$ such that $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d') = \mathbf{i}$, then $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d') = \mathbf{i}$ for every $d' \in (s')^{\tilde{J}}$;*
- *if $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d') = \mathbf{t}$, then $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d'') = \mathbf{f}$ for every $d'' \neq d'$;*
- *if $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d') = \mathbf{f}$, then there exists a d'' such that $\mathscr{G}_F^{\tilde{J}}(\overline{d}.d'') \neq \mathbf{f}$;*
- *if $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d') = \mathbf{u}$, then there exists a $d'' \neq d'$ such that $\mathscr{G}_F^{\tilde{J}}(\overline{d}.d'') = \mathbf{u}$.*

Proof. Define the Σ-structure $\tilde{I}$ by $P^{\tilde{I}} = P^{\tilde{J}}$ for every predicate symbol P and $F^{\tilde{I}}(\overline{d}) = \{d' \mid \mathscr{G}_F^{\tilde{J}}(\overline{d}, d') \in \{\mathbf{t}, \mathbf{u}\}\}$. It is straightforward to check that $\mathscr{G}(\tilde{I}) = \tilde{J}$. ■

2.2.3 Pairs of two-valued structures

Another way to represent a four-valued structure $\tilde{I}$ over a vocabulary Σ is by a pair of two-valued structures with the same domain. The two structures in such a pair represent what is certainly true, respectively false, in $\tilde{I}$.

Definition 2.4. Let $\tilde{I}$ be a four-valued structure over Σ. The *certainly true part* (ct-part) of $\tilde{I}$ is the two-valued structure $\tilde{I}^{\mathrm{ct}}$ over $\mathscr{G}(\Sigma)$ with the same domain as $\tilde{I}$, defined by

- $\overline{d} \in P^{\tilde{I}^{\mathrm{ct}}}$ iff $P^{\tilde{I}}(\overline{d}) \geq_p \mathbf{t}$ for every $P \in \Sigma_{\mathrm{pred}}$;
- $(\overline{d}, d') \in \mathscr{G}_F^{\tilde{I}^{\mathrm{ct}}}$ iff $F^{\tilde{I}}(\overline{d}) \subseteq \{d'\}$ for every $F \in \Sigma_{\mathrm{func}}$.

Similarly, the *certainly false part* of $\tilde{I}$ is the two-valued $\mathscr{G}(\Sigma)$-structure $\tilde{I}^{\mathrm{cf}}$ with the same domain as $\tilde{I}$, defined by

- $\overline{d} \in P^{\tilde{I}^{\mathrm{cf}}}$ iff $P^{\tilde{I}}(\overline{d}) \geq_p \mathbf{f}$ for every $P \in \Sigma_{\mathrm{pred}}$;
- $(\overline{d}, d') \in \mathscr{G}_F^{\tilde{I}^{\mathrm{cf}}}$ iff $d' \notin F^{\tilde{I}}(\overline{d})$ for every $F \in \Sigma_{\mathrm{func}}$.

For any pair (I, J) of two-valued $\mathscr{G}(\Sigma)$-structures, there exists a unique $\mathscr{G}(\Sigma)$-structure $\tilde{I}$ such that $I = \tilde{I}^{\mathrm{ct}}$ and $J = \tilde{I}^{\mathrm{cf}}$. If this structure $\tilde{I}$ is function consistent, then it follows from Proposition 2.6 that there exists a unique Σ-structure $\tilde{J}$ such that $I = \tilde{J}^{\mathrm{ct}}$ and $J = \tilde{J}^{\mathrm{cf}}$. It follows that for any Σ-structure $\tilde{I}$, $\tilde{I}^{\mathrm{ct}}$ and $\tilde{I}^{\mathrm{cf}}$ completely determine $\tilde{I}$. As such, we often introduce a four-valued structure $\tilde{I}$ by the pair $(\tilde{I}^{\mathrm{ct}}, \tilde{I}^{\mathrm{cf}})$. Also, we write $P^{\tilde{I}} = (P^{\tilde{I}^{\mathrm{ct}}}, P^{\tilde{I}^{\mathrm{cf}}})$ and may use, e.g., $(\{(2,3),(3,3)\}, \{(1,3),(9,5)\})$ to denote the value assigned to *ContainsShip* by the structure in Example 2.6.

Observe that if $\tilde{I}$ is three-valued, then $P^{\tilde{I}^{\mathrm{ct}}}$ and $P^{\tilde{I}^{\mathrm{cf}}}$ are disjoint for any predicate symbol $P \in \mathscr{G}(\Sigma)$. If $\tilde{I}$ is two-valued, then $P^{\tilde{I}^{\mathrm{ct}}}$ and $P^{\tilde{I}^{\mathrm{cf}}}$ are each others complement in $\mathfrak{s}(P)^{\tilde{I}}$. Also, if $\tilde{I} \leq_p \tilde{J}$, then $P^{\tilde{I}^{\mathrm{ct}}} \subseteq P^{\tilde{J}^{\mathrm{ct}}}$ and $P^{\tilde{I}^{\mathrm{cf}}} \subseteq P^{\tilde{J}^{\mathrm{cf}}}$ for every predicate symbol $P \in \mathscr{G}(\Sigma)_{\mathrm{pred}}$.

Instead of representing a Σ-structure $\tilde{I}$ by a pair of two-valued structures over $\mathscr{G}(\Sigma)$, we can as well represent it by one two-valued structure J over a vocabulary that contains a pair of predicates $(P^{\mathrm{ct}}, P^{\mathrm{cf}})$ for each predicate $P \in \mathscr{G}(\Sigma)$. The predicates P^{ct} are then interpreted in J by P's interpretation in $\tilde{I}^{\mathrm{ct}}$, the predicates P^{cf} by P's interpretation in $\tilde{I}^{\mathrm{cf}}$. Formally, denote by Σ^{tf} the vocabulary defined by $\Sigma^{\mathrm{tf}}_{\mathrm{sort}} = \Sigma_{\mathrm{sort}}$, $\Sigma^{\mathrm{tf}}_{\mathrm{func}} = \emptyset$ and

$$\begin{aligned}\Sigma^{\mathrm{tf}}_{\mathrm{pred}} =& \{P^{\mathrm{ct}}(\overline{s}) \mid P(\overline{s}) \in \Sigma_{\mathrm{pred}}\} \cup \{P^{\mathrm{cf}}(\overline{s}) \mid P(\overline{s}) \in \Sigma_{\mathrm{pred}}\} \\ & \cup \{\mathscr{G}_F^{\mathrm{ct}}(\overline{s}, s') \mid F(\overline{s}) : s' \in \Sigma_{\mathrm{func}}\} \cup \{\mathscr{G}_F^{\mathrm{cf}}(\overline{s}, s') \mid F(\overline{s}) : s' \in \Sigma_{\mathrm{func}}\}.\end{aligned}$$

That is, Σ^{tf} contains two predicates P^{ct} and P^{cf} for each predicate that occurs in Σ, and similarly for functions. If $\tilde{I}$ is a Σ-structure, then we denote by $\tilde{I}^{\mathrm{tf}}$ the two-valued Σ^{tf}-structure defined by $(P^{\mathrm{ct}})^{\tilde{I}^{\mathrm{tf}}} = P^{\tilde{I}^{\mathrm{ct}}}$ and $(P^{\mathrm{cf}})^{\tilde{I}^{\mathrm{tf}}} = P^{\tilde{I}^{\mathrm{cf}}}$. Thus, $\tilde{I}^{\mathrm{tf}}$ can be seen as a combination of the structures $\tilde{I}^{\mathrm{ct}}$ and $\tilde{I}^{\mathrm{cf}}$.

We now show that the value of a formula φ in a structure $\tilde{I}$ can be obtained by computing the value of two formulas φ^{ct} and φ^{cf} over Σ^{tf} in $\tilde{I}^{\mathrm{tf}}$. Define for a TNF sentence φ over Σ the sentences φ^{ct} and φ^{cf} over Σ^{tf} by simultaneous induction:

- $(P(\overline{x}))^{\mathrm{ct}} = P^{\mathrm{ct}}(\overline{x})$ and $(P(\overline{x}))^{\mathrm{cf}} = P^{\mathrm{cf}}(\overline{x})$;
- $(F(\overline{x}) = y)^{\mathrm{ct}} = \mathscr{G}_F^{\mathrm{ct}}(\overline{x}, y)$ and $(F(\overline{x}) = y)^{\mathrm{cf}} = \mathscr{G}_F^{\mathrm{cf}}(\overline{x}, y)$;
- $(\neg\varphi)^{\mathrm{ct}} = \varphi^{\mathrm{cf}}$ and $(\neg\varphi)^{\mathrm{cf}} = \varphi^{\mathrm{ct}}$;
- $(\varphi \wedge \psi)^{\mathrm{ct}} = \varphi^{\mathrm{ct}} \wedge \psi^{\mathrm{ct}}$ and $(\varphi \wedge \psi)^{\mathrm{cf}} = \varphi^{\mathrm{cf}} \vee \psi^{\mathrm{cf}}$;
- $(\varphi \vee \psi)^{\mathrm{ct}} = \varphi^{\mathrm{ct}} \vee \psi^{\mathrm{ct}}$ and $(\varphi \vee \psi)^{\mathrm{cf}} = \varphi^{\mathrm{cf}} \wedge \psi^{\mathrm{cf}}$;

- $(\forall x\ \varphi)^{\mathrm{ct}} = \forall x\ \varphi^{\mathrm{ct}}$ and $(\forall x\ \varphi)^{\mathrm{cf}} = \exists x\ \varphi^{\mathrm{cf}}$;
- $(\exists x\ \varphi)^{\mathrm{ct}} = \exists x\ \varphi^{\mathrm{ct}}$ and $(\exists x\ \varphi)^{\mathrm{cf}} = \forall x\ \varphi^{\mathrm{cf}}$.

The intuition is that φ^{ct} denotes a sentence that is true iff φ is certainly true while φ^{cf} is a sentence that is true iff φ is certainly false. This explains, e.g., the definition $(\neg\varphi)^{\mathrm{ct}} = \varphi^{\mathrm{cf}}$: $\neg\varphi$ is certainly true iff φ is certainly false. As another example, $(\varphi \wedge \psi)^{\mathrm{cf}} = \varphi^{\mathrm{cf}} \vee \psi^{\mathrm{cf}}$ states that $\varphi \wedge \psi$ is certainly false if φ or ψ is certainly false.
For a pair of formulas (φ_1, φ_2), a structure $\tilde{I}$ and variable assignment θ, we denote the pair of truth values $(\tilde{I}\theta(\varphi_1), \tilde{I}\theta(\varphi_2))$ by $\tilde{I}\theta(\varphi_1, \varphi_2)$. We identify the pairs $(\mathbf{t}, \mathbf{f})$, $(\mathbf{f}, \mathbf{t})$, $(\mathbf{f}, \mathbf{f})$ and $(\mathbf{t}, \mathbf{t})$ with, respectively, the truth values $\mathbf{t}$, $\mathbf{f}$, $\mathbf{u}$ and $\mathbf{i}$. Intuitively, the first value in the pairs states whether something is certainly true, the second value whether it is certainly false. It follows that, e.g., $(\mathbf{t}, \mathbf{f})$ corresponds to saying that something is certainly true and not certainly false and therefore identifies with $\mathbf{t}$. As another example, $(\mathbf{t}, \mathbf{t})$ states that something is both certainly true and false, which identifies with inconsistency. Using these equalities, the next proposition expresses that the value of a formula in a four-valued structure $\tilde{I}$ can be computed by evaluating φ^{ct} and φ^{cf} in the two-valued structure $\tilde{I}^{\mathrm{tf}}$.

Proposition 2.7. *Let φ be a TNF formula, $\tilde{I}$ a structure and θ a variable assignment. Then $\tilde{I}\theta(\varphi) = \tilde{I}^{\mathrm{tf}}\theta(\varphi^{\mathrm{ct}}, \varphi^{\mathrm{cf}})$.*

Proof. The proof is by structural induction. We prove one of the inductive cases.
As an example of an inductive case, let φ be the formula $\forall x\ \psi$. If $\tilde{I}\theta(\varphi) = \mathbf{u}$, then $\tilde{I}\theta[x/d](\psi) \in \{\mathbf{t}, \mathbf{u}\}$ for every $d \in \mathfrak{s}(x)^{\tilde{I}}$. Moreover, $\tilde{I}\theta[x/d](\psi) = \mathbf{u}$ for at least one d. From the induction hypothesis, it follows that $\tilde{I}^{\mathrm{tf}}\theta[x/d](\psi^{\mathrm{ct}}) = \mathbf{f}$ for at least one d, and $\tilde{I}^{\mathrm{tf}}\theta[x/d](\psi^{\mathrm{cf}}) = \mathbf{f}$ for every d. Therefore,

$$\tilde{I}^{\mathrm{tf}}\theta(\varphi^{\mathrm{ct}}, \varphi^{\mathrm{cf}}) = \tilde{I}^{\mathrm{tf}}\theta(\forall x\ \psi^{\mathrm{ct}}, \exists x\ \psi^{\mathrm{cf}}) = (\mathbf{f}, \mathbf{f}) = \mathbf{u}.$$

Vice versa, if $\tilde{I}^{\mathrm{tf}}\theta(\varphi^{\mathrm{ct}}, \varphi^{\mathrm{cf}}) = \mathbf{u}$, then $\tilde{I}^{\mathrm{tf}}\theta[x/d](\psi^{\mathrm{ct}}) = \mathbf{f}$ for at least one d, and $\tilde{I}^{\mathrm{tf}}\theta[x/d](\psi^{\mathrm{ct}}) = \mathbf{f}$ for every d. It follows from the induction hypothesis that $\tilde{I}\theta[x/d](\psi) \in \{\mathbf{t}, \mathbf{u}\}$ for every $d \in \mathfrak{s}(x)^{\tilde{I}}$ and that $\tilde{I}\theta[x/d](\psi) = \mathbf{u}$ for at least one d. We conclude that $\tilde{I}\theta(\varphi) = \mathbf{u}$.
All other inductive cases, and all base cases can be proven in a similar way. ■

It follows from Proposition 2.7 and Theorem 2.1 that $\tilde{I}\theta(\varphi)$ can be computed in polynomial time in $|\tilde{I}|$ and polynomial space in $|\varphi|$.
Another interesting property of the formulas φ^{ct} and φ^{cf} is stated in the following proposition.

Proposition 2.8. *For every formula φ, neither φ^{ct} nor φ^{cf} contain a negation symbol '$\neg$'.*

Proof. Follows directly from the definition of φ^{ct} and φ^{cf}. ■

Chapter 3

Extended first-order logic

Although FO is a very expressive logic, there are several concepts that cannot be expressed in it. A well-known example of such a concept is the transitive closure of a relation. Other concepts can be expressed in FO, but not in an easy, concise or natural manner. In this chapter we present the logic FO($\cdot$), an extension of FO designed to overcome some of these issues. Since FO($\cdot$) extends FO and has a clear formal and informal semantics, it is suitable for being used as the language underlying a KBS (Denecker and Vennekens, 2008).
Some of the extensions can be considered *syntactic sugar*: FO extended with these constructs is not more expressive than FO itself, but the extensions are convenient for modelling many real-life domains. In fact, we define the semantics of these extensions by specifying how they are reduced to plain FO. Partial functions, subsorts and arithmetic belong to this category. For other extensions, namely aggregates and inductive definitions, adding them to FO does lead to higher expressivity.
It has to be noted that FO($\cdot$) is a moving target. Future applications of knowledge base systems may reveal the need for additional extensions. At the end of this chapter we mention some extensions that are currently investigated. FO($\cdot$) as presented in this chapter is the language currently implemented in the finite model generator IDP (see Chapter 5).

3.1 Partial functions

In FO all functions are *total*, i.e., there is an image for every tuple in the domain of the function. On the contrary, several functions that occur in mathematics or in real-life applications are partial. For example, the integer division function $Div : \mathbb{Z} \to \mathbb{Z} : (x, y) \mapsto x/y$ is partial: tuples of the form $(x, 0)$ have no image under Div. The function *Spouse* is partial on the domain of all people. In a description of a battleship puzzle, we could have a function $ShipDir(Ship) : Direction$ mapping ships to their direction (horizontal or vertical). It makes sense to only define this function for ships with a length larger than one, thus

making *ShipDir* partial on the domain of all ships. In this section we extend FO with partial functions.

3.1.1 Ambiguous formulas

We denote the extension of FO with partial functions by FO(PF) . That is, an FO(PF) vocabulary Σ may contain partial function symbols $F(s_1, \ldots, s_n) : s_{n+1}$ and the interpretation of such a symbol in a Σ-structure I is a partial function $F^I : s_1^I \times \cdots \times s_n^I \to s_{n+1}^I$.
The semantics of FO(PF) is less straightforward, since arbitrary use of partial function symbols creates an ambiguity problem. Consider for instance the formula

$$\mathit{Woman}(\mathit{Spouse}(\mathit{John})). \tag{3.1}$$

When John indeed has a spouse, the meaning of this formula is clear: John's spouse is a woman. Problems arise when John has no spouse. The meaning of (3.1) becomes unclear since $\mathit{Spouse}(\mathit{John})$ then denotes a non-existing person. In principle, there are two different ways to interpret (3.1). Either it means "John has a spouse and his spouse is a woman", or it means "If John has a spouse, then his spouse is a woman". Clearly these interpretations of the formula are different. The former is false if John has no spouse, while the latter is true in the same situation. Formally, the ambiguity can be seen by rewriting the formula in a non-ambiguous form, using either the equivalence (2.13) or (2.14) of Section 2.1.3. The former yields the formula

$$\exists x\ (\mathit{Spouse}(\mathit{John}) = x \wedge \mathit{Woman}(x)), \tag{3.2}$$

while the latter produces

$$\forall x\ (\mathit{Spouse}(\mathit{John}) = x \Rightarrow \mathit{Woman}(x)). \tag{3.3}$$

Here, the atoms $\mathit{Spouse}(\mathit{John}) = x$ should be interpreted as $\mathscr{G}_{\mathit{Spouse}}(\mathit{John}, x)$. Note that these rewritings correspond to the two different informal readings of the formula we mentioned. The logical equivalence *in FO* of (3.2) and (3.3) reflects the fact that there is no ambiguity problem in case *Spouse* is a total function.
Several options to avoid the ambiguity have been proposed in the literature, amongst others by Frege (1892), Russell (1905) and Kleene (1952). The simplest solution is to restrict the syntax of formulas. One could, e.g., only allow terms of the form $F(\bar{t})$ in contexts where it is certain that the object denoted by $F(\bar{t})$ exists. This option is often taken in mathematics, where terms like, e.g., $\frac{1}{0}$ are considered nonsense. In the context of a KBS, this approach is too restrictive. For instance, in the area of planning, often a partial function like $\mathit{Do}(\mathit{Time}) : \mathit{Action}$ is used, assigning actions to certain time points. If one searches for a plan, the interpretation of *Do* is initially unknown and hence it is unknown where *Do* is defined. Consequently, *Do* cannot be used in sentences describing the constraints on a plan. Note also that the clearly non-ambiguous

formulas (3.2) and (3.3) are not allowed according to the restricted syntax, as long as it is not certain whether *John* is married.
Being a little bit less restrictive, one could allow atoms of the form $F(\bar{t}) = t'$ where F is a partial function, as long as $\bar{t}$ and t' do not contain partial functions. These atoms are then interpreted by $\mathscr{G}_F(\bar{t}, t')$. Formulas (3.2) and (3.3) are allowed according to this relaxed restriction.
An entirely different approach to avoid ambiguity was initiated by Kleene (1952), who solved the problem by moving to three-valued logic. If John has no spouse, $\mathit{Woman}(\mathit{Spouse}(\mathit{John}))$ would be considered unknown. However, the use of three-valued semantics in a KBS has disadvantages. The Kleene semantics (the one we presented in Section 2.2) is counterintuitive on some formulas. For instance, the formula $P \vee \neg P$ is unknown in every structure where P is unknown. Counterintuitive semantics are highly undesirable in a KBS. On the other hand, if *supervaluation* (see, e.g., van Fraassen, 1966) is used, $P \vee \neg P$ evaluates to true in every structure. However, supervaluation is computationally too expensive to be useful in a KBS.
Still another option is to define precisely in which cases an ambiguous atom like (3.1) is interpreted by (3.2) and in which cases by (3.3). This approach was taken by Russell (1905). Non-ambiguous atoms like $\mathit{Spouse}(\mathit{John}) = x$ remain interpreted by $\mathscr{G}_{\mathit{Spouse}}(\mathit{John}, x)$. When designing FO(PF), we opted for this option. It does not involve (complicated) three-valued semantics. It does not restrict the syntax and therefore allows an expert to write formulas involving partial functions in a concise way.[8] Finally, one can still express each possible intended two-valued semantics by writing non-ambiguous formulas.

3.1.2 Semantics of FO(PF)

We now formally define the semantics of partial functions in FO(PF). Call a term *ambiguous* if it is of the form $F(\bar{t})$, where F is a partial function. Call an atom *ambiguous* if it is of the form $P(\bar{t})$, $F(\bar{t}) = t'$ or $t' = F(\bar{t})$, and $\bar{t}$ contains an ambiguous term. Clearly, (3.1) is an ambiguous atom. If φ is an atom containing a term t and x is a variable not occurring in φ we denote by $\varphi^{\exists}_{t/x}$ the formula $\exists x\ (t = x \wedge \varphi[t/x])$ and by $\varphi^{\forall}_{t/x}$ the formula $\forall x\ (t = x \Rightarrow \varphi[t/x])$.
Our observation is that the intended interpretation of ambiguous atoms depends on the context where they occur. Usually, the intended interpretation is the cautious one, i.e., the one that maximizes the value of sentences of a theory according to the truth order.

Definition 3.1. A formula ψ occurs *positively* (negatively) in a formula φ if it occurs in the scope of an even (odd) number of negations in φ.

Definition 3.2. A *one-step rewrite* of a formula φ is a formula φ' obtained from φ by replacing an ambiguous atomic subformula ψ of φ with ambiguous term t by $\psi^{\forall}_{t/x}$ or $\psi^{\exists}_{t/x}$. A one-step rewrite is *cautious* if ψ occurs positively and

[8] In an actual implementation using FO(PF), it is of course a good idea to produce a warning message when a user writes an ambiguous formula.

is replaced by $\psi^{\forall}_{t/x}$, or ψ occurs negatively and is replaced by $\psi^{\exists}_{t/x}$. A *rewrite* of φ is a formula φ' obtained by starting from φ and applying one-step rewrites until no ambiguous atomic subformulas are left. A rewrite is *cautious* if all one-step rewrites applied to obtain φ' are cautious.

Example 3.1. Assuming that *Spouse* is a partial function, the only cautious rewrite of the sentence

$$\forall x\ (Male(Spouse(x)) \Rightarrow Female(x)) \tag{3.4}$$

is given by

$$\forall x\ (\exists y\ (Spouse(x) = y \wedge Male(y)) \Rightarrow Female(x)).$$

The meaning of this sentence is that a person with a male spouse is female. On the contrary, the only non-cautious rewrite of (3.4) has the meaning "If all spouses of a person are male, then that person is female". In particular this would mean that any person without a spouse is female.

Example 3.2. Assuming F and C are partial functions, then the sentence $P(F(C))$ has the following two cautious rewrites:

$$\forall x\ (\exists y\ (C = y \wedge F(y) = x) \Rightarrow P(x)), \tag{3.5}$$
$$\forall y\ (C = y \Rightarrow \forall x\ (F(y) = x \Rightarrow P(x))). \tag{3.6}$$

The first one is obtained by first substituting $F(C)$ by x and then C by y. The second one is obtained by first substituting C by y and then $F(y)$ by x.

A important observation is that (3.5) and (3.6) are equivalent in the sense that if I is a $\mathscr{G}(\Sigma)$ structure, then $I \models \mathscr{G}((3.5))$ iff $I \models \mathscr{G}((3.6))$. This property is true in general.

Proposition 3.1. *If φ_1 and φ_2 are two cautious rewrites of a formula φ over Σ and I is a $\mathscr{G}(\Sigma)$-structure, then $I \models \mathscr{G}(\varphi_1)$ iff $I \models \mathscr{G}(\varphi_2)$.*

Proof. See Appendix A. ■

Proposition 3.1 implies that the following definition makes sense.

Definition 3.3. Let Σ be an FO(PF) vocabulary, I a Σ-structure, θ a variable mapping and φ an formula over Σ. We say that $I\theta \models \varphi$ if $\mathscr{G}(I)\theta \models \mathscr{G}(\psi)$, where ψ is a cautious rewrite of φ.

As mentioned, this semantics maximizes the truth of formulas. This is formally stated in the following proposition.

Proposition 3.2. *Let φ_1 and φ_2 be two rewrites of a formula φ over Σ and let I be a $\mathscr{G}(\Sigma)$ structure. If φ_1 is a cautious rewrite and $I\theta \models \mathscr{G}(\varphi_2)$, then $I\theta \models \mathscr{G}(\varphi_1)$.*

Proof. See Appendix A. ■

3.2 Interrelated sorts

In mathematics and in many real-life domains outside mathematics, classes of objects are often interrelated in the sense that objects can belong to multiple classes. For example, sparrows belong to the class of birds, to the class of flying animals, to the class of winged animals, etc. In a convenient FO-based knowledge representation language, one should be able to express such relations between sorts. However, this cannot be done in many-sorted FO, as all sorts denote disjoint sets of objects.[9] In this section, we adapt FO to include interrelated sorts. As we will see, the resulting logic necessarily includes partial functions. We denote this logic by FO(SUB).

The development of logics with more general sort systems than many-sorted FO was initiated by Oberschelp (1962). In particular, Oberschelp developed several *order-sorted logics*, based on the observation that concepts are often ordered in a hierarchy. For example, $\mathbb{N} \subsetneq \mathbb{Z} \subsetneq \mathbb{Q} \subsetneq \mathbb{R}$. Emperor penguins are penguins, penguins are birds, birds are animals. Order-sorted logics allow to represent such hierarchies in a natural way. We explain the simplest order-sorted logic, called *S-Logic*. In S-Logic, a vocabulary contains besides the usual symbols also a partial ordering $\subseteq$ on its set of sorts. A structure I for such a vocabulary Σ needs to satisfy the extra requirement that $s_1^I \subseteq s_2^I$ for each two sorts $s_1, s_2 \in \Sigma_{\text{sort}}$ such that $s_1 \subseteq s_2$. A formula in order-sorted logic is well-sorted: for every function symbol $F(s_1, \ldots, s_n) : s_{n+1} \in \Sigma_{\text{func}}$, terms of the form $F(t_1, \ldots, t_n)$ are only allowed iff $\mathfrak{s}(t_1) \subseteq s_1$, ..., $\mathfrak{s}(t_n) \subseteq s_n$, and for every predicate symbol $P(s_1, \ldots, s_n)$, atoms of the form $P(t_1, \ldots, t_n)$ are allowed iff $\mathfrak{s}(t_1) \subseteq s_1$, ..., $\mathfrak{s}(t_n) \subseteq s_n$.

As recognized by many authors, including Oberschelp himself (1989), S-Logic is not the most general order-sorted logic and for a natural modelling of many domains, it is too restrictive.

Example 3.3. Consider a vocabulary with sorts Human, $\mathit{Man} \subseteq \mathit{Human}$ and $\mathit{Woman} \subseteq \mathit{Human}$, and three predicate symbols $\mathit{Parent}(\mathit{Human}, \mathit{Human})$, $\mathit{Father}(\mathit{Man}, \mathit{Human})$ and $\mathit{Mother}(\mathit{Woman}, \mathit{Human})$. A natural modelling of the statement "Person x is the parent of person y iff x is the father or mother of y" is then given by

$$\forall x \forall y \ (\mathit{Parent}(x, y) \Leftrightarrow \mathit{Father}(x, y) \vee \mathit{Mother}(x, y)). \tag{3.7}$$

According to S-Logic, this sentence is not well-sorted. Yet, it is clearly not nonsensical.

3.2.1 Base and subsorts

We propose a less restrictive order-sorted logic, denoted by FO(SUB). In this logic, there is a distinction between *base sorts* (or *categories*) and *subsorts*. The

[9] Of course, it is possible to say that two objects belonging to different sorts, say sort Bird and Animal, are equal by adding a predicate $\mathit{Equal}(\mathit{Animal}, \mathit{Bird})$ and interpreting *Equal(a,b)* by *Animal a is the same creature as bird b*. However, writing $a = b$ for an animal a and bird b remains a syntax error.

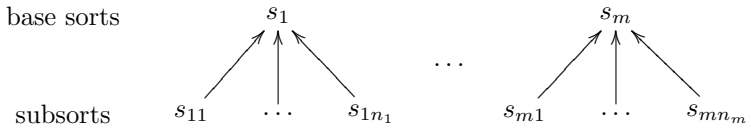


Figure 3.1: A sort hierarchy in FO(SUB)

former sorts serve to rule out ill-sorted formulas such as $x = y$ when x and y have incomparable sorts, e.g., *Human* and *Colour*. That is, the base sorts are used to avoid 'category errors'. The subsorts on the other hand can be used to represent hierarchies.
Formally, a vocabulary Σ in FO(SUB) consists of

- A set Σ_{sort} of sorts, partitioned into two subsets Σ_{base} and Σ_{sub}. Sorts in the former subset are called *base sorts* of Σ, sorts in the latter are called *subsorts*.
- A function $\mathfrak{base} : \Sigma_{\text{sort}} \to \Sigma_{\text{base}}$ such that $\mathfrak{base}(s) = s$ for any $s \in \Sigma_{\text{base}}$.
- A set Σ_{pred} of predicate symbols with associated sorts, containing at least for each $s \in \Sigma_{\text{sort}}$ a unary predicate s_{pred} with sort s. We call s_{pred} a *sort predicate*.
- A set Σ_{func} of function symbols and a set Σ_{var} of variable symbols, with associated sorts.

A Σ-structure I for an FO(SUB) vocabulary Σ assigns, as usual, appropriate values to the sorts, predicate and function symbols of Σ and moreover satisfies the following constraints.

- For each s_1 and s_2 in Σ_{sort}, if $\mathfrak{base}(s_1) = s_2$, then $s_1^I \subseteq s_2^I$.
- For each sort predicate s_{pred}, $s_{\text{pred}}^I = s^I$.

A term t in FO(SUB) is *well-sorted* if t is a variable, or t is of the form $F(t_1, \ldots, t_n)$ such that $t_1, \ldots, t_n$ are well-sorted terms, $\mathfrak{s}(F) = (s_1, \ldots, s_n)$ and $\mathfrak{base}(\mathfrak{s}(t_i)) = \mathfrak{base}(s_i)$ for $1 \leq i \leq n$. Likewise, an atomic formula $t_1 = t_2$ is well-sorted if t_1 and t_2 are well-sorted and $\mathfrak{base}(\mathfrak{s}(t_1)) = \mathfrak{base}(\mathfrak{s}(t_2))$. An atomic formula $P(t_1, \ldots, t_n)$ is well-sorted if $t_1, \ldots, t_n$ are well-sorted, $\mathfrak{s}(P) = (s_1, \ldots, s_n)$ and $\mathfrak{base}(\mathfrak{s}(t_i)) = \mathfrak{base}(s_i)$ for $1 \leq i \leq n$. An FO(SUB) theory may only contain well-sorted formulas.
Observe that only very simple sort hierarchies can be built using the $\mathfrak{base}$ function. Indeed, all hierarchies have the structure shown in Figure 3.1. More complex relations between sorts can be expressed by (extended) FO sentences over the sort predicates. In particular, all sort hierarchies that can be defined in S-Logic can be expressed using sort predicates.

Example 3.4. Consider a vocabulary Σ where $\Sigma_{\text{base}} = \{\mathit{Animal}\}$,

$$\Sigma_{\text{sub}} = \{\mathit{Bird}, \mathit{FlyingAnimal}, \mathit{FlyingBird}\},$$

and $\mathfrak{base}$ is the function mapping each sort to Animal. This expresses that the objects of sorts Bird, $\mathit{FlyingAnimal}$ and $\mathit{FlyingBird}$ also belong to Animal. We can express that the flying birds are a subset of the birds by the sentence $\forall x\ (\mathit{FlyingBird}_{\text{pred}}(x) \Rightarrow \mathit{Bird}_{\text{pred}}(x))$. The sentence

$$\forall x\ (\mathit{FlyingBird}_{\text{pred}}(x) \Leftrightarrow (\mathit{Bird}_{\text{pred}}(x) \wedge \mathit{FlyingAnimal}_{\text{pred}}(x)))$$

expresses that the set of flying birds is precisely the intersection of the set of birds and the set of flying animals.

There are two difficulties when defining the semantics of FO(SUB). First, for a structure I, variable assignment θ, function symbol $F(s_1, \ldots, s_n) : s_{n+1}$ and well-sorted term $F(t_1, \ldots, t_n)$, the value $I\theta(F(t_1, \ldots, t_n))$ does not necessarily exist. Indeed, it is possible that $I\theta(t_i) \notin s_i^I$ since it is not required anymore that $\mathfrak{s}(t_i) = s_i$. We solve this issue by viewing F as a partial function of sort $(\mathfrak{base}(s_1), \ldots, \mathfrak{base}(s_n), s_{n+1})$. Secondly, for a predicate $P(s_1, \ldots, s_n)$ and well-sorted atom $P(t_1, \ldots, t_n)$, it could be that $I\theta(t_1) \notin s_i^I$ for some $1 \leq i \leq n$. We define $I\theta \not\models P(t_1, \ldots, t_n)$ in this case. These two difficulties being solved, we define the semantics of FO(SUB) in exactly the same manner as the semantics for FO(PF): for a formula φ, $I\theta \models \varphi$ if $\mathscr{G}(I)\theta \models \mathscr{G}(\psi)$, where ψ is a cautious rewrite of φ.

Example 3.5. In FO(SUB), the sorts in the vocabulary of Example 3.3 are partitioned in $\Sigma_{\text{base}} = \{\mathit{Human}\}$ and $\Sigma_{\text{sub}} = \{\mathit{Man}, \mathit{Woman}\}$. The $\mathfrak{base}$ function maps Man and Woman to Human. The semantics of sentence (3.7) is then as desired: for any structure I that satisfies (3.7) and variable assignment θ such that $\theta(x) = d_x$, $\theta(y) = d_y$ and $(d_x, d_y) \in \mathit{Parent}^I$, we have that $(d_x, d_y) \in \mathit{Father}^I$ or $(d_x, d_y) \in \mathit{Woman}^I$. The former can only be the case if $d_y \in \mathit{Man}^I$, the latter if $d_y \in \mathit{Woman}^I$.

Example 3.6. Let Σ be the vocabulary defined by $\Sigma_{\text{base}} = \{\mathit{Animal}, \mathit{Int}\}$, $\Sigma_{\text{sub}} = \{\mathit{FlyingAnimal}, \mathit{Insect}\}$, $\Sigma_{\text{pred}} = \{\mathit{Large}(\mathit{Int}), \mathit{Dangerous}(\mathit{Animal})\}$ and $\Sigma_{\text{func}} = \{\mathit{Wingspan}(\mathit{FlyingAnimal}) : \mathit{Int}\}$. If i is a variable of sort Insect, then

$$\forall i\ (\mathit{Large}(\mathit{Wingspan}(i)) \Rightarrow \mathit{Dangerous}(i)) \tag{3.8}$$

expresses that every insect that flies and has a large wingspan is dangerous. Indeed, since $\mathit{Wingspan}$ is a partial function, formula (3.8) is interpreted by

$$\forall i\ (\exists w\ (\mathscr{G}_{\mathit{Wingspan}}(i, w) \wedge \mathit{Large}(w)) \Rightarrow \mathit{Dangerous}),$$

and the condition $\mathscr{G}_{\mathit{Wingspan}}(i, w) \wedge \mathit{Large}(w)$ can only be satisfied if i belongs to sort $\mathit{FlyingAnimal}$.

3.2.2 Reducing order-sorted logic to many-sorted logic

FO(SUB) can be reduced to many-sorted logic. Let T be an FO(SUB) theory over vocabulary Σ and let I be a Σ-structure. Denote by Σ' the vocabulary defined by

$$\begin{aligned}
\Sigma'_{\text{sort}} &= \Sigma_{\text{sort}} \setminus \Sigma_{\text{sub}} \\
\Sigma'_{\text{pred}} &= \{P(\mathfrak{base}(\overline{s})) \mid P(\overline{s}) \in \Sigma_{\text{pred}}\} \\
\Sigma'_{\text{func}} &= \{F(\mathfrak{base}(\overline{s})) : \mathfrak{base}(s') \mid F(\overline{s}) : s' \in \Sigma_{\text{func}}\} \\
\Sigma'_{\text{var}} &= \{x : \mathfrak{base}(s) \mid x : s \in \Sigma_{\text{var}}\}
\end{aligned}$$

Note that Σ' is a many-sorted vocabulary, not an order-sorted one. There exists a theory T' over Σ' such that there is a one-to-one correspondence between the models of T and T'. Indeed, let T' be the theory obtained from T by recursively replacing every quantified subformula $(\forall x\ \varphi)$ by $(\forall x\ (\mathfrak{s}(x)_{\text{pred}}(x) \Rightarrow \varphi))$ and every quantified subformula $(\exists x\ \varphi)$ by $(\exists x\ (\mathfrak{s}(x)_{\text{pred}}(x) \wedge \varphi))$ and adding the following sentences:

- The sentences $\forall x\ (s_{\text{pred}}(x) \Rightarrow \mathfrak{base}(s)_{\text{pred}}(x))$ and $\exists x\ s_{\text{pred}}(x)$ for every subsort $s \in \Sigma_{\text{sub}}$;
- The sentence $\forall x_1 \cdots \forall x_n\ (P(x_1, \ldots, x_n) \Rightarrow (s_{1\,\text{pred}}(x_1) \wedge \ldots \wedge s_{n\,\text{pred}}(x_n)))$ for every predicate $P(s_1, \ldots, s_n) \in \Sigma_{\text{pred}}$;
- The sentence $\forall x_1 \cdots \forall x_n \forall y\ (F(x_1, \ldots, x_n) = y \Rightarrow (s_{1\,\text{pred}}(x_1) \wedge \ldots \wedge s_{n\,\text{pred}}(x_n) \wedge s_{n+1\,\text{pred}}(y)))$ for every function symbol $F(s_1, \ldots, s_n) : s_{n+1} \in \Sigma_{\text{func}}$.

One can check that for every M' of the theory T' the Σ-structure M that assigns $s^M = s^{M'}_{\text{pred}}$ for every subsort $s \in \Sigma_{\text{sub}}$ and corresponds to M' on all other symbols is a model of T. Note that the extra sentences added in T' ensure that M is indeed a Σ-structure. Vice versa, if $M \models T$, then $M|_{\Sigma'} \models T'$. We conclude that FO(SUB) can be reduced to FO(PF).
In a similar way, many-sorted logic can be reduced to one-sorted logic. It suffices to add one base sort to a many-sorted vocabulary and making all other sorts subsort of this new base sort. Then exactly the same construction as above transforms many-sorted logic into one-sorted logic. Hence, from a theoretical perspective, many-sorted logic does not add to one-sorted logic. In a practical system however it is very convenient to use many-sorted or order-sorted logic. It allows for much more concise theories and to detect some errors more easily. For example, if a user forgets a sort predicate and writes in one-sorted logic the sentence $\exists x\ \mathit{Father}(x, \mathit{John})$ instead of $\exists x\ (\mathit{Man}_{\text{pred}}(x) \wedge \mathit{Father}(x, \mathit{John}))$, a system may find models where the father of *John* is a woman.[10] In order-sorted logic, the same error cannot occur.

[10] Such errors occur in practice. For example, one of the errors in the second ASP competition (Denecker et al., 2009) was due to the use of a one-sorted system. See the example *EdgeMatching* on `www.cs.kuleuven.be/~dtai/events/ASP-competition/ERROR/`.

3.3 Integer arithmetic

Now that we have extended FO with partial functions and subsorts, it is straightforward to add, e.g., integer arithmetic. It is obvious that the modelling of many real-life domains will involve integer arithmetic. We denote the extension of FO(SUB) with integer arithmetic by FO(SU,IA). In this logic, every vocabulary contains at least base sort $\mathfrak{Int}$, predicate symbol $\leq (\mathfrak{Int}, \mathfrak{Int})$, function symbols '+', '−', '·', '/', 'mod' with sort $(\mathfrak{Int}, \mathfrak{Int}) : \mathfrak{Int}$, function symbol 'abs' with sort $(\mathfrak{Int}) : \mathfrak{Int}$ and for each $a \in \mathbb{Z}$ a constant a with sort $\mathfrak{Int}$. In each FO(SU,IA) structure I, $\mathfrak{Int}^I = \mathbb{Z}$, $+$ is interpreted by integer addition, $-$ by subtraction, etc. Observe that '/' and 'mod' denote partial functions.

Example 3.7. In the battleship puzzle, we can describe the concept of adjacent positions by a predicate $\mathit{Adjacent}(\mathit{Row}, \mathit{Col}, \mathit{Row}, \mathit{Col})$ with intended meaning that $\mathit{Adjacent}(r_1, c_1, r_2, c_2)$ is true if square (r_1, c_1) is adjacent or equal to square (r_2, c_2). If *Row* and *Col* are subsorts of $\mathfrak{Int}$, i.e., $\mathfrak{base}(\mathit{Row}) = \mathfrak{base}(\mathit{Col}) = \mathfrak{Int}$, then we can define *Adjacent* by

$$\begin{aligned}\forall r_1 \forall c_1 \forall r_2 \forall c_2 \ (&\mathit{Adjacent}(r_1, c_1, r_2, c_2) \\ &\Leftrightarrow (\mathrm{abs}(r_2 - r_1) \leq 1) \wedge (\mathrm{abs}(c_2 - c_1) \leq 1)).\end{aligned}$$

In the same way, we could add, e.g., real arithmetic.
In the rest of this thesis, we call a FO(SU,IA) structure I *finite* if it is finite on all symbols that are not "built-in". That is, I assigns finite domains to every sort besides $\mathfrak{Int}$, finite relations to every predicate symbol, except $\leq (\mathfrak{Int}, \mathfrak{Int})$ and for every (partial) function symbol F, the set

$$\{\overline{d} \mid \text{there exists a } d' \text{ such that } F^I(\overline{d}) = d'\}$$

is finite, except for the functions '+', '−', etc. Note that if a vocabulary Σ contains a total function symbol $F(\overline{s}) : s'$ such that $\mathfrak{Int} \in \overline{s}$, every Σ-structure is necessarily infinite. We call a four-valued FO(SU,IA) structure $\tilde{I}$ finite if $\tilde{I}^{\mathrm{tf}}$ is finite.

3.4 Aggregates

Aggregates are functions that have a set or multi-set as argument. An example is the function CARD returning the cardinality of a set. Not all aggregates are total functions. An example of a partial aggregate function is the function MAX defining the maximum of the values in a set of integers: for the empty set, the maximum is undefined.
In many cases, using aggregates allows for more succinct theories (Simons et al., 2002), and often faster inference (Faber et al., 2008).

Example 3.8. To model the rules of a battleship puzzle, we have to express that on each row or column, the number of positions that contain a ship matches exactly the number associated to that row or column. To this

end, we extend the vocabulary of Example 2.2 with a new predicate symbol $\mathit{ContainsShip}(\mathit{Row}, \mathit{Col})$ with the intended meaning that (r, c) belongs to the relation denoted by $\mathit{ContainsShip}$ iff (r, c) represents a position that contains part of a ship.
Now the mentioned rules for rows and columns in a battleship puzzle are expressed by the following sentences containing aggregates. Here, r and c are variables of sort Row, respectively Col.

$$\forall r \ (\text{CARD}\{c \mid \mathit{ContainsShip}(r, c)\} = \mathit{RowNumber}(r)) . \tag{3.9}$$

$$\forall c \ (\text{CARD}\{r \mid \mathit{ContainsShip}(r, c)\} = \mathit{ColNumber}(c)) . \tag{3.10}$$

Sentence (3.9) is read as "For each row r, the number of columns c such that (r, c) contains part of a ship is equal to the number associated to r". Sentence (3.10) expresses the same constraint for columns.
Modelling these constraints in FO(SU,IA) is more involved. We illustrate this by giving a possible way to represent the constraint for rows. First introduce three auxiliary functions $\mathit{AuxNumRow}(\mathit{Row}, \mathit{Col}) : \mathit{Num}$, $\mathit{MinCol} : \mathit{Col}$ and $\mathit{MaxCol} : \mathit{Col}$. Then express that MinCol and MaxCol represent the minimum, respectively maximum column of the grid:

$$\forall c \ (c \geq \mathit{MinCol} \wedge c \leq \mathit{MaxCol}). \tag{3.11}$$

Next, state that $\mathit{AuxNumRow}(r, c) = n$ if the number of parts of ships on row r on columns between the minimum column and c is equal to n:

$$\begin{aligned}
&\forall r \ (\mathit{AuxNumRow}(r, \mathit{MinCol}) = 1 \Leftrightarrow \mathit{ContainsShip}(r, \mathit{MinCol})),\\
&\forall r \ (\mathit{AuxNumRow}(r, \mathit{MinCol}) = 0 \Leftrightarrow \neg \mathit{ContainsShip}(r, \mathit{MinCol})),\\
&\forall r \forall c \ (c < \mathit{MaxCol} \Rightarrow\\
&\quad (\mathit{AuxNumRow}(r, c+1) = \mathit{AuxNumRow}(r, c) + 1 \Leftrightarrow \mathit{ContainsShip}(r, c+1))),\\
&\forall r \forall c \ (c < \mathit{MaxCol} \Rightarrow\\
&\quad (\mathit{AuxNumRow}(r, c+1) = \mathit{AuxNumRow}(r, c) \Leftrightarrow \neg \mathit{ContainsShip}(r, c+1))).
\end{aligned}$$

Finally, the constraint we wanted to model is expressed by

$$\forall r \ (\mathit{AuxNumRow}(r, \mathit{MaxCol}) = \mathit{RowNumber}(r)).$$

Clearly, the modelling using aggregates is more concise and more intuitive. Moreover, the modelling without aggregates is not correct in case of a battleship puzzle with an infinite grid. Indeed, if Col^I is infinite in a structure I, then there exists no value MaxCol such that (3.11) is satisfied in I.

We denote the extension of FO(SU,IA) with aggregates by FO(AGG). A *set expression* in FO(AGG) is an expression of the form

$$\{\overline{x} \mid \varphi\}, \tag{3.12}$$

where $\overline{x}$ is a tuple of variables and φ a formula. We use $\{(t_1, \varphi_1), (t_2, \varphi_2), \ldots, (t_n, \varphi_n)\}$, where $t_1, \ldots, t_n$ are terms of the same base sort and $\varphi_1, \ldots, \varphi_n$ are formulas, as a shorthand for the set expression

$$\{(x, y) \mid (x = t_1 \wedge y = 1 \wedge \varphi_1) \vee \ldots \vee (x = t_n \wedge y = n \wedge \varphi_n)\}.$$

We use $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ as a shorthand for $\{(1, \varphi_1), \ldots, (1, \varphi_n)\}$. A set expression $\{(x_1, \ldots, x_n) \mid \varphi\}$ is called an *integer set expression* if $n > 0$ and $\mathfrak{base}(\mathfrak{s}(x_1)) = \mathfrak{Int}$.
The value of a set expression V in a structure I under variable assignment θ is a set $I\theta(V)$ defined by $I\theta(\{\overline{x} \mid \varphi\}) = \{\overline{d} \mid I\theta[\overline{x}/\overline{d}] \models \varphi\}$. Observe that for an integer set expression V, $I\theta(V)$ is a set of tuples $(a_1, \ldots, a_n)$ where $a_1 \in \mathbb{Z}$.

Example 3.9. The value of set expression $\{c \mid \mathit{ContainsShip}(r, c)\}$ (see formula (3.9)) in a structure I under variable assignment θ is the set of all columns c such that square $(\theta(r), c)$ contains part of a ship. For example, if I describes the solved battleship puzzle shown in Figure 2.2 and $\theta(r) = 3$, then $I\theta(\{c \mid \mathit{ContainsShip}(r, c)\}) = \{3, 5, 6, 7, 9\}$.

Example 3.10. If $\theta(y)$ denotes a person who has fever and a low blood pressure, but no low heart rate according to structure I, then

$$I\theta(\{HasFever(y), Lowheartrate(y), LowBloodPressure(y)\}) = \{(1, 1), (1, 3)\}.$$

In this text, we consider the aggregate function symbols CARD, SUM, PROD, MIN and MAX.[11] An *aggregate term* is an expression of the form CARD(V_1), SUM(V_2), PROD(V_2), MIN(V_2) or MAX(V_2), where V_1 is a set expression and V_2 an integer set expression. Each of these terms has sort $\mathfrak{Int}$.
Each of the aggregate functions symbols denotes a partial function that is total on the class of all finite non-empty sets. Let I be a structure and θ a variable assignment. If $I\theta(V)$ is finite, we define

- $I\theta(\text{CARD}(V))$ is the cardinality of the set $I\theta(V)$;
- $I\theta(\text{SUM}(V)) = \sum_{(a_1,\ldots,a_n)\in I\theta(V)}(a_1)$ if $I\theta(V) \neq \emptyset$ and $I\theta(\text{SUM}(V)) = 0$ otherwise;
- $I\theta(\text{PROD}(V)) = \prod_{(a_1,\ldots,a_n)\in I\theta(V)}(a_1)$ if $I\theta(V) \neq \emptyset$ and $I\theta(\text{PROD}(V)) = 1$ otherwise.

If $I\theta(V)$ is finite and non-empty, we define

- $I\theta(\text{MIN}(V)) = \min\{a_1 \mid (a_1, \ldots, a_n) \in I\theta(V)\}$;
- $I\theta(\text{MAX}(V)) = \max\{a_1 \mid (a_1, \ldots, a_n) \in I\theta(V)\}$.

[11] Other aggregate function symbols can be added in the same manner. The ones we present are those that are currently implemented in IDP.

The semantics of FO(AGG) formulas is given by their cautious rewrite, treating all aggregate atoms containing $\text{F}(V)$ as ambiguous, except for atoms of the form $\text{F}(V) \leq t$, $\text{F}(V) \geq t$, $\text{F}(V) < t$, $\text{F}(V) > t$ or $\text{F}(V) = t$, where t does not contain aggregates or partial functions. If $I\theta(V)$ is infinite, then we define $I\theta \not\models \varphi$ for every non-ambiguous atom that contains $I\theta(V)$. If $I\theta(V) = \emptyset$, $I\theta(\text{MIN}(V))$ is treated as $+\infty$ and $I\theta(\text{MAX}(V))$ as $-\infty$. That is, if $I\theta(V) = \emptyset$, then we define $I\theta \models \text{MIN}(V) \geq t$, $I\theta \not\models \text{MIN}(V) \leq t$, etc.
To define the semantics of partial functions that occur inside aggregate terms, we extend the notion of positive and negative occurrence of a formula to FO(AGG). To this end, we treat each direct subformula of a set expression as a new entity, occurring positively in any context.

Definition 3.4. A formula ψ occurs *positively* in a formula φ if either it does not occur inside an aggregate term in φ and it occurs in the scope of an even number of negations, or it occurs positively inside a direct subformula of an aggregate term in φ. A formula ψ occurs *negatively* in φ if it occurs in φ, but not positively.

3.5 Inductive definitions

One of the famous examples of concepts that are not expressible in FO is the concept of reachability in a graph. That is, there is no FO formula φ over the vocabulary consisting of two predicates $\mathit{Edge}/2$ and $\mathit{Reach}/2$ such that in any model M of φ, $(d_1, d_2) \in \mathit{Reach}^M$ iff there is a path from d_1 to d_2 in the graph represented by Edge^M. In fact, it is fair to say that no property of graphs that requires recursion is expressible in FO (Grädel et al., 2007). Reachability is such a property. A simple inductive (recursive) definition of reachability is given by:

- d_2 is reachable from d_1 if there is an edge between d_1 and d_2.
- d_2 is reachable from d_1 if there is some intermediate node d such that d is reachable from d_1 and d_2 is reachable from d.

Although they cannot be expressed in FO, inductively definable concepts have applications in many real-life computational problems such as automated planning or problems involving dynamic systems (Denecker and Ternovska, 2007, 2008).
In this section, we present FO(ID) (Denecker, 2000; Denecker and Ternovska, 2008), the extension of FO with a construct to represent some of the most common types of inductive definitions: monotone induction, induction over a well-founded order and iterated inductive definitions. A monotone inductive definition consists of rules that state objects can be added to the defined set given the *presence* of certain other objects of the set. The definition of reachability given above is an example of a monotone inductive definition. An induction over a well-founded order states that objects d can be added to the defined set given the *presence* or *absence* of certain objects in the defined set that are strictly less than d in according to a given well-founded order. Definitions by

structural induction, such as the definition of the satisfaction relation '$\models$' are inductive definitions over a well-founded order. In the case of '$\models$', the given well-founded order is the subformula order. Iterated inductive definitions (Buchholz et al., 1981) are a generalization of monotone definitions and definitions over a well-founded order.
A theory in FO(ID) has the appearance of an FO theory augmented with a collection of *logic programs*. As illustrated by Denecker and Ternovska (2008), this entails that definitions in FO(ID) can not only be used to represent mathematical concepts, but also for the sort of common sense knowledge that is often represented by logic programs, such as (local forms of) the closed world assumption, inheritance, exceptions, defaults, causality, etc.

3.5.1 Syntax and semantics

A *definition* Δ is a finite set of rules of the form

$$\forall \overline{x}\ (P(\bar{t}) \leftarrow \varphi[\overline{y}]),$$

where P is a predicate, φ a formula, $\overline{y} \subseteq \overline{x}$ and $\bar{t}$ is a tuple of terms with free variables among $\overline{x}$. $P(\bar{t})$ is called the *head* and φ the *body* of the rule. The connective '$\leftarrow$' is called *definitional implication* and is to be distinguished from the connective '$\Rightarrow$'. Predicates that appear in the head of a rule of Δ are called *defined predicates* of Δ. The set of all defined predicates of Δ is denoted $\mathrm{Def}(\Delta)$. All other symbols are called *open* with respect to Δ. The set of all open symbols of Δ is denoted $\mathrm{Open}(\Delta)$. Hence $\mathrm{Open}(\Delta) = (\Sigma_{\mathrm{pred}} \setminus \mathrm{Def}(\Delta)) \cup \Sigma_{\mathrm{func}}$.

Example 3.11. The following definition defines the predicate *Reach* in terms of open predicate *Edge*.

$$\left\{ \begin{array}{ll} \forall x \forall y\ (\mathit{Reach}(x,y) & \leftarrow \mathit{Edge}(x,y)), \\ \forall x \forall y\ (\mathit{Reach}(x,y) & \leftarrow \exists z\ (\mathit{Reach}(x,z) \wedge \mathit{Reach}(z,y))) \end{array} \right\} \tag{3.13}$$

Informally, this definition expresses that y can be reached from x in the graph represented by *Edge*, if either there is an edge between x and y, i.e., $\mathit{Edge}(x,y)$ is true, or if there is some intermediate node z such that z can be reached from x and y can be reached from z.

The formal semantics of definitions is given by their well-founded model (Van Gelder et al., 1991). We borrow the presentation of this semantics from Denecker and Vennekens (2007).

Definition 3.5. Let Δ be a definition and $\tilde{I}$ a three-valued structure. A *well-founded induction for* Δ *extending* $\tilde{I}$ is a (possibly transfinite) sequence $\langle \tilde{J}_\xi \rangle_{0 \leq \xi \leq \alpha}$ of three-valued structures such that

1. $\tilde{J}_0$ assigns $P^{\tilde{J}_0}(\overline{d}) = \mathbf{u}$, if P is a defined predicate and corresponds to $\tilde{I}$ on the open symbols;
2. For each limit ordinal $\lambda \leq \alpha$, $\tilde{J}_\lambda = \mathrm{lub}_{\leq_p}\{\tilde{J}_\xi \mid \xi < \lambda\}$;

3. For every ordinal ξ, $\tilde{J}_{\xi+1}$ relates to $\tilde{J}_\xi$ in one of the following ways:

   (a) $\tilde{J}_{\xi+1} = \tilde{J}_\xi[V/\mathbf{t}]$, where V is a set of domain atoms such that for each $P(\overline{d}) \in V$, $P^{\tilde{J}_\xi}(\overline{d}) = \mathbf{u}$ and there exists a rule $\forall \overline{x}\ (P(\overline{t}) \leftarrow \varphi)$ in Δ and a tuple of domain elements $\overline{d}'$ such that $[\overline{x}/\overline{d}'](\overline{t}) = \overline{d}$ and $\tilde{J}_\xi[\overline{x}/\overline{d}'](\varphi) = \mathbf{t}$.

   (b) $\tilde{J}_{\xi+1} = \tilde{J}_\xi[U/\mathbf{f}]$, where U is a set of domain atoms, such that for each $P(\overline{d}) \in U$, $P^{\tilde{J}_\xi}(\overline{d}) = \mathbf{u}$ and for all rules $\forall \overline{x}\ (P(\overline{t}) \leftarrow \varphi)$ in Δ and tuples $\overline{d}'$ such that $[\overline{x}/\overline{d}'] = \overline{d}$, $\tilde{J}_{\xi+1}[\overline{x}/\overline{d}'](\varphi) = \mathbf{f}$.

Intuitively, (3a) says that domain atoms $P(\overline{d})$ can be made true if there is a rule with $P(\overline{t})$ in the head and body φ such that φ is already true, given a variable assignment that interprets $\overline{t}$ by $\overline{d}$. On the other hand (3b) explains that $P(\overline{d})$ can be made false if there is no possibility of making a corresponding body true, except by circular reasoning. The set U, called an *unfounded set*, is a witness to this: making all atoms in U false also makes all corresponding bodies false.

A well-founded induction is called *terminal* if it cannot be extended anymore. The limit of a terminal well-founded induction is its last element. Denecker and Vennekens (2007) show that each terminal well-founded induction for Δ extending $\tilde{I}$ has the same limit, which corresponds to the well-founded model of Δ extending $\tilde{I}|_{\mathrm{Open}(\Delta)}$. The well-founded model is denoted by $\mathrm{wfm}_\Delta(\tilde{I})$. In general, $\mathrm{wfm}_\Delta(\tilde{I})$ is three-valued.

A two-valued structure I satisfies definition Δ, denoted $I \models \Delta$, if $I = \mathrm{wfm}_\Delta(I)$. The extension of FO with inductive definition is denoted by FO(ID). An FO(ID) theory is a set of FO sentences and definitions.[12] A two-valued structure satisfies an FO(ID) theory T if it satisfies every sentence and every definition of T.

Example 3.12. Consider a structure I for vocabulary

$$\langle \{ \mathit{Vtx} \}, \{ \mathit{Edge}(\mathit{Vtx}, \mathit{Vtx}), \mathit{Reach}(\mathit{Vtx}, \mathit{Vtx}) \}, \emptyset \rangle$$

such that $\mathit{Vtx}^I = \{a, b, c\}$ and Edge^I represents the graph

$$a \longrightarrow b \rightleftarrows c\ .$$

If I satisfies the definition (3.13), then $(d_1, d_2) \in \mathit{Reach}^I$ iff there is a path from d_1 to d_2 in the graph above. Indeed, $\langle \tilde{J}_i \rangle_{0 \le i \le 2}$ is a well-founded induction for

[12] One could also allow definitions inside formulas (Denecker, 2000), but we are not aware of any real-life application where this more general version of FO(ID) is necessary.

definition (3.13) extending I if, e.g.,[13]

$$\begin{aligned}
Reach^{\tilde{J}_0} &= (\emptyset, \emptyset)\\
Reach^{\tilde{J}_1} &= (\{(a,b),(b,c),(c,b)\}, \emptyset)\\
Reach^{\tilde{J}_2} &= (\{(a,b),(b,c),(c,b),(a,c),(b,b),(c,c)\}, \emptyset)\\
Reach^{\tilde{J}_2} &= (\{(a,b),(b,c),(c,b),(a,c),(b,b),(c,c)\}, \{(a,a),(b,a),(c,a)\})
\end{aligned}$$

Example 3.13. Let T be the theory

$$\begin{array}{l}
\forall v_1 \forall v_2 \ (Path(v_1, v_2) \Rightarrow Edge(v_1, v_2)),\\
\forall v_1 \forall v_2 \forall v_3 \ (Path(v_1, v_2) \wedge Path(v_1, v_3) \Rightarrow v_2 = v_3),\\
\forall v_1 \forall v_2 \forall v_3 \ (Path(v_1, v_3) \wedge Path(v_2, v_3) \Rightarrow v_1 = v_2),\\
\forall v \ \neg Path(v, Start),\\
\forall v \ Reach(v),\\
\left\{ \begin{array}{l} \forall v \ (Reach(v) \leftarrow v = Start), \\ \forall v \ (Reach(v) \leftarrow \exists w \ (Reach(w) \wedge Path(w, v))) \end{array} \right\}.
\end{array}$$

If M is a model of T, then the edges in $Path^I$ form a Hamiltonian path starting at $Start^I$ in the graph represented by $Edge^I$.

For the extensions of FO described in the previous sections, it is straightforward to see that the informal semantics of formulas coincide with the formal semantics.[14] For inductive definitions however, this is less obvious. When trying to figure out which objects belong to a recursively defined concept in a mathematical text, a reader takes into account the order in which the rules of the definition need to be applied. For instance, because the satisfaction relation '$\models$' is defined by structural induction, i.e., according to the subformula order, it is first checked whether the subformulas φ and ψ of formula $\varphi \wedge \psi$ are satisfied in a structure before one can conclude whether $\varphi \wedge \psi$ itself is satisfied in that structure. However, inductive definitions in FO(ID) do not come with an explicit order to evaluate the rules, neither does the formal semantics, i.e., the definition of well-founded model, fix such an order. Denecker and Ternovska (2008) showed that the informal and formal semantics of FO(ID) nevertheless coincide. Basically, the argument is that, e.g., in the case of constructing the set of objects defined by an induction over a well-founded order, the crucial aspect of the given well-founded order is the following: it delays deciding whether an object is in (out) the defined set until it is certain that the condition of the rule

[13] Recall that we use the notation (V_1, V_2) to introduce a four-valued interpretation of a predicate P: V_1 lists the tuples that certainly belong to P, V_2 the tuples that certainly do not belong to P.

[14] At least, if there is indeed a non-ambiguous intuitive reading of the formula. When the occurrence of partial functions in a formula leads to several non-equivalent readings, we can of course not say that the informal and formal semantics of that formula coincide. A similar remark can be made for non-total definitions (see below) since these definitions also do not have a clear intuitive reading.

that allows such a decision is true (false) and cannot become false (true) anymore by later applications of rules. This is precisely what is also accomplished by well-founded model semantics: if during the construction of a well-founded induction, a defined domain atom becomes true, this is because its rule body is already true. Since further applying rules, i.e., extending the well-founded induction, produces more and more precise structures, a rule body that becomes true can never become false anymore. Vice versa, if a defined atom becomes false in a well-founded induction, this is because no application of the rules can make any rule body defining that atom true. Indeed, assume towards a contradiction that the set U of domain atoms is an unfounded set in the nth structure $\tilde{I}_n$ of a well-founded induction $\langle \tilde{I}_i \rangle_{0 \leq i \leq k}$ and that m is the lowest number larger than n, such that there exists a rule $\forall \overline{x}\ (P(\overline{x}) \leftarrow \varphi)$ and tuple $\overline{d}$ where $P(\overline{d}) \in U$ and $\tilde{I}_m[\overline{x}/\overline{d}](\varphi) = \mathbf{t}$. Then $\tilde{I}_m(A) \leq_t \mathbf{u}$ for every $A \in U$. As such $\tilde{I}_m[U/\mathbf{f}] \geq_p \tilde{I}_m$ and therefore $\tilde{I}_m[U/\mathbf{f}][\overline{x}/\overline{d}](\varphi) = \mathbf{t}$. But since U is an unfounded set at step n and $P(\overline{d}) \in U$, $\tilde{I}_m[U/\mathbf{f}][\overline{x}/\overline{d}](\varphi) \geq_p \tilde{I}_n[U/\mathbf{f}][\overline{x}/\overline{d}](\varphi) = \mathbf{f}$. This is a contradiction. We refer to the work of Denecker and Ternovska (2008) for a more in-depth argumentation.

3.5.2 Classes of definitions

A class of definitions that occur frequently in practice are positive definitions.

Definition 3.6. A definition Δ is *positive* if none of the defined predicates of Δ occurs negatively in the rule bodies of Δ.

Definition (3.13) is an example of a positive definition.

Definition 3.7. A formula φ over Σ is *monotone with respect to subvocabulary* σ *of* Σ if $\tilde{I}(\varphi) \leq_t \tilde{J}(\varphi)$ for every pair of Σ-structures $\tilde{I} \leq_t \tilde{J}$ that are two-valued on $\Sigma \setminus \sigma$. A definition Δ is monotone if every rule body in Δ is monotone with respect to $\mathrm{Def}(\Delta)$.

Clearly, every positive definition is monotone. The well-founded model of a monotone definition can be obtained by first applying step 3a of Definition 3.5 until a fixpoint $\tilde{I}$ is reached. The set of all domain atoms that are unknown in $\tilde{I}$ forms an unfounded set, and hence these atoms can be made false to obtain a terminal well-founded induction. This strategy was used in Example 3.12.
Another class of definitions are *stratified definitions*.

Definition 3.8. A definition Δ is *stratified* if there exists a function l from $\mathrm{Def}(\Delta)$ to $\mathbb{N}$ such that if P and Q are two defined predicates of Δ and P occurs negatively in a rule of Δ defining Q, $l(P) < l(Q)$.

In mathematics, (inductive) definitions need to be well-formed. That is, the definition of a mathematical property P should express for each possible object whether it has property P or not. If for some object a it is unknown whether a has property P according to the definition of P, then P is not well-defined. The same holds if the definition expresses that a cannot have property P and

cannot not have property P. For instance, defining P by the definition "for every a, a has property P if a does not have property P" is clearly not allowed in mathematics. Well-formed definitions in FO(ID) are called *total definitions*.

Definition 3.9. A definition is called *total* if for any $\Sigma|_{\mathrm{Open}(\Delta)}$-structure J the well-founded model of Δ extending J is two-valued. A definition Δ is *total on a class $\mathcal{I}$ of* Open(Δ)*-structures* if for each structure $I \in \tilde{I}$, the well-founded model of Δ extending I is two-valued.

Total definitions correspond to well-formed definitions since they define for every defined predicate P and for each tuple of domain elements whether $\overline{d}$ belongs to the relation denoted by P or not. In practice, all definitions that occur in FO(ID) theories are total. For example, this is the case for all FO(ID) theories used in the second ASP competition (Denecker et al., 2009). In general, checking whether a definition is total is undecidable (Schlipf, 1995). However, there are several broad and easily recognizable classes of total definitions. For example, all monotone and stratified definitions are total (Van Gelder et al., 1991).

3.5.3 Rewriting definitions

In this section, we review some rules that can be applied to rewrite definitions to a suitable normal form. Such a normal form may facilitate the presentation of theoretical results and the implementation of practical systems.

Moving functions outside heads

Every definition Δ can be rewritten to an equivalent definition Δ' such that no function symbols occur in the head of rules of Δ'. Indeed, we can define Δ' by

$$\begin{aligned}\Delta' =&\{\forall x_1 \cdots \forall x_n \forall \overline{y}\ (P(x_1, \ldots, x_n) \leftarrow x_1 = t1 \wedge \ldots \wedge x_n = t_n \wedge \varphi)\\ &\mid \forall \overline{y}\ (P(t_1, \ldots, t_n) \leftarrow \varphi) \in \Delta\}.\end{aligned}$$

Merging rules

Every definition Δ can be rewritten to an equivalent definition Δ' where each predicate of Def(Δ') is defined by exactly one rule. For every defined predicate P, the rule defining P in Δ' is the disjunction of all rules defining P in Δ.

Proposition 3.3. *Let Δ be a definition such that no function symbols occur in the head of rules of Δ and let Δ' be the definition that contains for every $P \in \mathrm{Def}(\Delta)$ the rule*

$$\forall \overline{x}\ (P(\overline{x}) \leftarrow ((\overline{x} = \overline{y}_1 \wedge \varphi_1) \vee \ldots \vee (\overline{x} = \overline{y}_n \wedge \varphi_n))),$$

where $\forall \overline{y}_1\ (P(\overline{y}_1) \leftarrow \varphi_1)$, ..., $\forall \overline{y}_n\ (P(\overline{y}_n) \leftarrow \varphi_n)$ are all the rules of Δ with P in the head. Then Δ and Δ' are logically equivalent.

Proof. The proposition is a direct consequence of the definition of well-founded induction: it is easy to check that each well-founded induction for Δ is one for Δ' and vice versa. ■

Merging and splitting definitions

Mariën et al. (2004) introduce a method to merge all definitions of a theory T over Σ into one Σ-equivalent definition. The method consists of introducing a new predicate P'/n for each predicate P/n that is defined in a definition of T. For each of these new predicates, the sentence $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow P'(\overline{x}))$ is added to the theory. Finally, all definitions $\Delta_1, \ldots, \Delta_m$ of T are replaced by the single definition $\bigcup_{1 \leq i \leq m} \Delta_i'$. Here, for every $1 \leq i \leq m$, Δ_i' is the definition that contains for every rule $\forall \overline{x}\ (P(\bar{t}) \leftarrow \varphi) \in \Delta_i$ the rule $\forall \overline{x}\ (P'(\bar{t}) \leftarrow \varphi')$, where φ' is obtained from φ by replacing every $Q \in \mathrm{Def}(\Delta)$ by Q'. Mariën et al. (2004) show that the resulting theory is Σ-equivalent to the original one. In practice it is often not necessary to introduce a new predicate P' for *every* defined predicate of T. In Appendix C, we illustrate this by sketching how definitions are merged in the IDP system.
The inverse operation, namely splitting a large definition into several smaller ones, was extensively studied by Vennekens et al. (2006) and Denecker and Ternovska (2008). It is beyond the scope of this text to review their results.

Predicate introduction

In Section 2.1.3 we introduced predicate introduction for FO: replacing a complex subformula ψ by an atom and defining the atom in terms of ψ. Predicate introduction for FO(ID) was investigated by Vennekens et al. (2007). The following theorem is their main result.

Theorem 3.4 (Vennekens et al. 2007)**.** *Let Δ be a definition over Σ and let $\varphi[\overline{x}]$ be a subformula with n free variables that occurs positively in a body of a rule in Δ. Let P/n be a new predicate and δ a monotone definition over a vocabulary $\Sigma' \supseteq (\Sigma \cup \{P\})$ such that $P \in \mathrm{Def}(\delta)$, $\mathrm{Def}(\delta) \cap \Sigma_{\mathrm{pred}} = \emptyset$ and $\mathrm{wfm}_\delta(\tilde{I})[\overline{x}/\overline{d}](P(\overline{x})) = \mathrm{wfm}_\delta(\tilde{I})[\overline{x}/\overline{d}](\varphi[\overline{x}])$ for every Σ'-structure $\tilde{I}$ and tuple of domain elements $\overline{d}$. Then Δ and $\Delta' \cup \delta$ are Σ-equivalent, where Δ' is the result of replacing $\varphi[\overline{x}]$ in Δ by $P(\overline{x})$.*

Theorem 3.4 ensures that predicate introduction inside a definition is possible, as long as only positively occurring subformulas are replaced, and the introduced predicates are defined by a rule inside the definition.

Corollary 3.5. *If Δ is a definition over Σ, $\varphi[\overline{x}]$ a subformula with n free variables that occurs positively in the body of a rule of Δ and P/n is a new predicate, then Δ is Σ-equivalent to $\Delta' \cup \{\forall \overline{x}\ (P(\overline{x}) \leftarrow \varphi[\overline{x}])\}$, where Δ' is the result of replacing $\varphi[\overline{x}]$ in Δ by $P(\overline{x})$.*

Completion of a definition

A well-known concept that we will use later on in this thesis is the *completion* of a definition Δ, which is an FO theory that is weaker than Δ.

Definition 3.10. The *completion* of a definition Δ is the FO theory that contains for every $P \in \mathrm{Def}(\Delta)$ the sentence

$$\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow ((\overline{x} = \overline{y}_1 \wedge \varphi_1) \vee \ldots \vee (\overline{x} = \overline{y}_n \wedge \varphi_n))),$$

where $\forall \overline{y}_1\ (P(\overline{y}_1) \leftarrow \varphi_1)$, $\ldots$, $\forall \overline{y}_n\ (P(\overline{y}_n) \leftarrow \varphi_n)$ are the rules in Δ with P in the head.

We denote the completion of Δ by $\mathrm{Comp}(\Delta)$. Clearly, every body of a rule in Δ occurs in $\mathrm{Comp}(\Delta)$. If T is a theory then we denote by $\mathrm{Comp}(T)$ the result of replacing in T all definitions by their completion. The following result states that the completion of T is weaker than T.

Theorem 3.6 (Denecker and Ternovska 2008)**.** $\Delta \models \mathrm{Comp}(\Delta)$ *and* $T \models \mathrm{Comp}(T)$ *for every definition* Δ *and* FO(ID) *theory* T*.*

3.6 Combining all extensions

In this section, we combine inductive definitions with all other presented extensions of FO. We denote the resulting logic by FO($\cdot$).

3.6.1 Combining partial functions and definitions

First we consider the extension of FO(ID) with partial functions. As before, it is sufficient to indicate which of the two rewriting formulas (2.13) or (2.14) are to be applied in which context. Definitions add two new contexts: heads and bodies of rules. We define how ambiguous atoms are interpreted in these contexts.

- Let $\forall \overline{x}\ (P(\overline{t}) \leftarrow \varphi)$ be a rule such that $P(\overline{t})$ contains an ambiguous term $F(\overline{t}')$. Then the interpretation of this rule is defined as the interpretation of the rule $\forall \overline{x} \forall y\ (P(\overline{t})[F(\overline{t}')/y] \leftarrow \varphi \wedge y = F(\overline{t}'))$, where y is a new variable.
- A rule $\forall \overline{x}\ (P(\overline{t}) \leftarrow \varphi)$ where φ contains ambiguous atoms is interpreted by $\forall \overline{x}\ (P(\overline{t}) \leftarrow \psi)$ where ψ is a rewrite of φ, obtained by only applying non-cautious one-step rewrites. Equivalently, ψ is the negation of a cautious rewrite of $\neg\varphi$.

It follows from Proposition 3.2 that this semantics minimizes the truth of defined atoms.

Example 3.14. Definition

$$\{\forall x\ (\mathit{FiscallyDependent}(x, \mathit{Spouse}(x)) \leftarrow \neg\, \mathit{Works}(x))\}$$

is interpreted by

$$\{\forall x \forall y\ (\mathit{FiscallyDependent}(x, y) \leftarrow \neg\, \mathit{Works}(x) \wedge \mathscr{G}_{\mathit{Spouse}}(x, y))\}.$$

Definition

$$\left\{ \begin{array}{ll} \forall x \forall y\ (Grandparent(x,y) & \leftarrow Parent(x, Father(y))), \\ \forall x \forall y\ (Grandparent(x,y) & \leftarrow Parent(x, Mother(y))) \end{array} \right\}$$

is interpreted by

$$\left\{ \begin{array}{ll} \forall x \forall y\ (Grandparent(x,y) & \leftarrow \exists z\ (Parent(x,z) \wedge \mathscr{G}_{Father}(y,z))), \\ \forall x \forall y\ (Grandparent(x,y) & \leftarrow \exists z\ (Parent(x,z) \wedge \mathscr{G}_{Mother}(y,z))) \end{array} \right\}.$$

3.6.2 Combining subsorts and definitions

When subsorts are added to FO(ID) it can be the case that a rule of a definition has a head of the form $P(x)$ such that neither $\mathfrak{s}(x) = \mathfrak{s}(P)$ nor $\mathfrak{base}(\mathfrak{s}(x)) = \mathfrak{s}(P)$. This can have undesired effects, as illustrated by the following example.

Example 3.15. Assume we want to define the even and odd numbers on a finite interval $[0, n]$ of the natural numbers. To this end, we introduce a sort s with $\mathfrak{base}(s) = \mathfrak{Int}$ that denotes the finite interval and two predicate symbols $Even(s)$ and $Odd(s)$. However, the definition

$$\left\{ \begin{array}{rl} Even(0) & \leftarrow \top, \\ Even(x+1) & \leftarrow Odd(x), \\ Odd(x+1) & \leftarrow Even(x) \end{array} \right\}$$

is problematic in any structure that interprets s by a finite interval $[0, n]$. Indeed, if n is an even number, then for any model M of the definition $M \models Even(n)$ and by the third rule of the definition also $M \models Odd(n+1)$. Yet, the semantics introduced in Section 3.2 enforced $M \not\models Odd(n+1)$ since $n+1$ does not belong to $\mathfrak{s}(Odd)^M$. A similar contradiction arises when n is an odd number.

To avoid these unexpected side-effects, we define the semantics of order-sorted FO(ID) such that for any structure I, a definition can never force a domain atom $\overline{d} \in P^I$ if $\overline{d} \notin \mathfrak{s}(P)^I$.

Definition 3.11. For any definition Δ in order-sorted FO(ID) and any structure I, $I \models \Delta$ if $I \models \Delta'$, where Δ' is obtained from Δ by replacing every rule $\forall \overline{x}\ (P(t_1, \ldots, t_n) \leftarrow \varphi) \in \Delta$ by $\forall \overline{x}\ (P(t_1, \ldots, t_n) \leftarrow \varphi \wedge s_{1\,\mathrm{pred}}(t_1) \wedge \ldots \wedge s_{n\,\mathrm{pred}}(t_n))$ where $\mathfrak{s}(P) = (s_1, \ldots, s_n)$.

Example 3.16. The definition of Example 3.15 is interpreted by

$$\left\{ \begin{array}{rl} Even(0) & \leftarrow s_{\mathrm{pred}}(0), \\ Even(x+1) & \leftarrow Odd(x) \wedge s_{\mathrm{pred}}(x+1), \\ Odd(x+1) & \leftarrow Even(x) \wedge s_{\mathrm{pred}}(x+1) \end{array} \right\}.$$

In a model M of this definition where s^M is a finite interval $[0, n]$, $Even^M$ and Odd^M are exactly the even and odd numbers in the finite interval $[0, n]$.

3.6.3 Aggregates inside definitions

The semantics of aggregates occurring inside the rules of a definition was extensively studied by Pelov et al. (2007). The basic idea is to extend the definition of the value $\tilde{I}\theta(\varphi)$ of an FO formula φ in a three-valued structure $\tilde{I}$ under θ to FO(AGG) formulas. If this definition is chosen such that for any two structures $\tilde{I}$ and $\tilde{J}$, $\tilde{I} \leq_p \tilde{J}$ implies $\tilde{I}\theta(\varphi) \leq_p \tilde{J}\theta(\varphi)$, then the definition of well-founded induction simply extends to the case where rule bodies contains aggregates. Pelov et al. (2007) provide an overview of different possibilities to define $\tilde{I}\theta(\varphi)$. We present one of them.

A *three-valued set* is a set where each element is annotated with one of the truth values $\mathbf{t}, \mathbf{f}$ or $\mathbf{u}$. We denote the annotations by superscripts. A three-valued set $\tilde{V}$ approximates a class of sets, namely all sets that certainly contain the elements of V annotated by $\mathbf{t}$ and possibly some of the elements of V annotated by $\mathbf{u}$. For example, $\{a^{\mathbf{t}}, b^{\mathbf{f}}, c^{\mathbf{u}}\}$ denotes a three-valued set, approximating the sets $\{a\}$ and $\{a, b\}$. We write $V \sqsubset \tilde{V}$ to denote that V is a set approximated by the three-valued set $\tilde{V}$.

In a three-valued structure, a set expression evaluates to a three-valued set.

Definition 3.12. The *value* $\tilde{I}\theta(V)$ of an FO set expression in three-valued structure $\tilde{I}$ under variable assignment θ is defined by

- $\tilde{I}\theta(\{\overline{x} \mid \varphi\}) = \{\overline{d}^{\mathbf{v}} \mid \tilde{I}\theta[\overline{x}/\overline{d}](\varphi) = \mathbf{v}\}$;
- $\tilde{I}\theta(\{(t_1, \varphi_1), (t_2, \varphi_2), \ldots, (t_n, \varphi_n)\}) = \{(I\theta(t_i), i)^{\mathbf{v}} \mid I\theta(\varphi_i) = \mathbf{v}\}$.

Example 3.17. Let $\tilde{I}$ be the three-valued structure that represents the unsolved battleship puzzle in Figure 2.1 (see Example 2.6). The value of the set expression $\{r \mid \mathit{ContainsShip}(r, 3)\}$ in $\tilde{I}$ is the set

$$\{1^{\mathbf{f}}, 2^{\mathbf{t}}, 3^{\mathbf{t}}, 4^{\mathbf{u}}, 5^{\mathbf{u}}, 6^{\mathbf{u}}, 7^{\mathbf{u}}, 8^{\mathbf{u}}, 9^{\mathbf{u}}, 10^{\mathbf{u}}\}. \tag{3.14}$$

Definition 3.13. The *minimal value* $\tilde{I}\theta(\mathrm{F}(V))_{\min}$ of an FO(AGG) term $\mathrm{F}(V)$ in structure $\tilde{I}$ under variable assignment θ is defined by

$$\tilde{I}\theta(\mathrm{F}(V))_{\min} = \min\{n \mid n = \mathrm{F}(v) \text{ for some } v \sqsubset \tilde{I}\theta(V)\},$$

if $\mathrm{F}(v)$ is defined for at least one $v \sqsubset \tilde{I}\theta(V)$. Similarly, the *maximal value* of $\mathrm{F}(V)$ is defined by

$$\tilde{I}\theta(\mathrm{F}(V))_{\max} = \max\{n \mid n = \mathrm{F}(v) \text{ for some } v \sqsubset \tilde{I}\theta(V)\},$$

if $\mathrm{F}(v)$ is defined for at least one $v \sqsubset \tilde{I}\theta(V)$.

Example 3.18 (Example 3.17 ctd.)**.** If $\tilde{I}$ is the structure defined in Example 2.6, then $\tilde{I}(\mathrm{CARD}(\{r \mid \mathit{ContainsShip}(r, 3)\}))_{\min} = 2$. Indeed, the smallest set approximated by the set (3.14) is the set $\{2, 3\}$. Likewise, one can check that $\tilde{I}(\mathrm{CARD}(\{r \mid \mathit{ContainsShip}(r, 3)\}))_{\max} = 9$.

Proposition 3.7 (Pelov et al. 2007)**.** *If* $\mathrm{F} \in \{\mathrm{CARD}, \mathrm{SUM}, \mathrm{PROD}, \mathrm{MIN}, \mathrm{MAX}\}$*, then* $\tilde{I}\theta(\mathrm{F}(V))_{\min}$ *and* $\tilde{I}\theta(\mathrm{F}(V))_{\max}$ *are computable in time polynomial in* $|\tilde{I}|$.

Several times in this text, we will facilitate the presentation by assuming, without loss of generality, that FO(·) theories are in term normal form (TNF). As in the case of FO, this normal form is needed to define the value of an FO(·) formula in a three-valued structure. The definition of TNF for FO(·) theories extends the one for FO theories (Definition 2.1).

Definition 3.14. An FO(·) theory T is in *term normal form* (TNF) if the following conditions are met:

- All negations in T occur directly in front of atoms;
- All atomic subformulas of T are of the form $P(\overline{x})$, $F(\overline{x}) = y$ or $z_1 = z_2$, where $\overline{x}$ is a tuple of variables and y, z_1 and z_2 are variables;
- All aggregate expressions in T are of the form $\text{F}(V) = x$, $\text{F}(V) \leq x$ or $\text{F}(V) \geq x$, where x is a variable.

Every theory can be rewritten to an equivalent theory in TNF. We indicated a rewriting strategy for FO theories below Definition 2.1. By applying equivalences (2.13) or (2.14), every aggregate expression can be brought in one of the forms $\text{F}(V) \leq x$, $\text{F}(V) \geq x$ or $\text{F}(V) = x$. Finally, the formulas $\neg(\text{F}(V) \leq x)$, $\neg(\text{F}(V) \geq x)$ and $\neg(\text{F}(V) = x)$ can be replaced by respectively $\text{F}(V) \geq x + 1$, $\text{F}(V) \leq x - 1$ and $((\text{F}(V) \geq x + 1) \vee (\text{F}(V) \leq x - 1))$.

Definition 3.15. The value of an FO(AGG) formula φ in TNF in structure $\tilde{I}$ under variable assignment θ is defined by adding the following cases to the definition of the value of an FO formula.

- $\tilde{I}\theta(\text{F}(V) \leq x) = \begin{cases} \mathbf{t} & \tilde{I}\theta(\text{F}(V))_{\max} \leq \theta(x) \\ \mathbf{u} & \text{if } \tilde{I}\theta(\text{F}(V))_{\min} \leq \theta(x) \text{ and } \tilde{I}\theta(\text{F}(V))_{\max} > \theta(x) \\ \mathbf{f} & \text{otherwise.} \end{cases}$
- $\tilde{I}\theta(x \leq \text{F}(V)) = \begin{cases} \mathbf{t} & \tilde{I}\theta(\text{F}(V))_{\min} \geq \theta(x) \\ \mathbf{u} & \text{if } \tilde{I}\theta(\text{F}(V))_{\min} < \theta(x) \text{ and } \tilde{I}\theta(\text{F}(V))_{\max} \geq \theta(x) \\ \mathbf{f} & \text{otherwise.} \end{cases}$
- $\tilde{I}\theta(\text{F}(V) = x) = \tilde{I}\theta(\text{F}(V) \leq x \wedge x \leq \text{F}(V))$.

Pelov et al. (2007) illustrate that this definition of the value of FO(AGG) formulas in three-valued structures is sufficiently precise for most examples found in the literature.

Example 3.19 (Example 3.17 ctd.)**.** The formula $\mathit{ColNumber}(3) = \text{CARD}\{r \mid \mathit{ContainsShip}(r, 3)\}$ can be rewritten to the equivalent TNF formula

$$\forall n \ (\mathit{ColNumber}(3) = n \Rightarrow \text{CARD}\{r \mid \mathit{ContainsShip}(r, 3)\} = n).$$

If $\tilde{I}$ is the structure of Example 2.6, then

$$\begin{aligned}
&\tilde{I}(\forall n\ (\mathit{ColNumber}(3) = n \Rightarrow \text{CARD}\{r \mid \mathit{ContainsShip}(r,3)\} = n)) \\
&= \tilde{I}[n/2](\text{CARD}\{r \mid \mathit{ContainsShip}(r,3)\} = n) \\
&= \text{glb}_{\leq_t} \left\{ \begin{array}{l} \tilde{I}[n/2](\text{CARD}\{r \mid \mathit{ContainsShip}(r,3)\} \leq n), \\ \tilde{I}[n/2](\text{CARD}\{r \mid \mathit{ContainsShip}(r,3)\} \geq n) \end{array} \right\} \\
&= \text{glb}_{\leq_t}\{\mathbf{t}, \mathbf{u}\} \\
&= \mathbf{u}
\end{aligned}$$

The third equation follows from the results in Example 3.18.

3.6.4 Defining functions

The construct for representing inductive definition can easily be extended to allow besides definitions of predicates also definitions of functions. It suffices to allow atoms of the form $F(\bar{t}) = t'$ in the head of rules in a definition. If Δ is a definition that defines functions, a two-valued structure I satisfies Δ if $\mathscr{G}(I) \models \Delta'$, where Δ' is obtained by replacing every rule of the form $\forall \overline{x}\ (F(\bar{t}) = t' \leftarrow \varphi)$ by $\forall \overline{x}\ (\mathscr{G}_F(\bar{t}, t') \leftarrow \varphi)$.

3.7 Final example

As a final example illustrating many of the constructs in FO($\cdot$), we describe the constraints that are satisfied by every solution for a battleship puzzle.

Example 3.20. Let Σ be the vocabulary of Example 2.6, augmented with the sort *Direction*, the predicate symbols *Adjacent*(*Row*, *Col*, *Row*, *Col*) and *ContainsShipS*(*Row*, *Col*, *Ship*) and function symbols *Horizontal* : *Direction*, *Vertical* : *Direction*, *InitRow*(*Ship*) : *Row*, *InitCol*(*Ship*) : *Col* and *ShipDir*(*Ship*) : *Direction*. The intended meaning of these symbols is as follows: *Adjacent*(r_1, c_1, r_2, c_2) means that (r_1, c_1) and (r_2, c_2) are adjacent or equal squares in the grid, *ContainsShipS*(r, c, s) means that (r, c) contains a part of ship s. *InitRow* and *InitCol* map a ship to the row and column of its first part, *ShipDir* maps a ship to its direction (horizontal or vertical). *Row*, *Col*, *Number* and *Length* are subsorts of $\mathfrak{Int}$. The following theory then expresses

all constraints a solution for a battleship puzzle should satisfy.

$$\left\{ \begin{array}{l} \forall c \forall s\ (\mathit{ContainsShipS}(\mathit{InitRow}(s), c, s) \leftarrow \\ \quad \mathit{ShipDir}(s) = \mathit{Horizontal} \\ \quad \wedge\ \mathit{InitCol}(s) \leq c \\ \quad \wedge\ c < \mathit{InitCol}(s) + \mathit{ShipLength}(s)), \\ \forall r \forall s\ (\mathit{ContainsShipS}(r, \mathit{InitCol}(c), s) \leftarrow \\ \quad \mathit{ShipDir}(s) = \mathit{Vertical} \\ \quad \wedge \mathit{InitRow}(s) \leq r \\ \quad r < \mathit{InitRow}(s) + \mathit{ShipLength}(s)) \end{array} \right\}, \tag{3.15}$$

$$\left\{ \begin{array}{l} \forall r \forall c\ (\mathit{ContainsShip}(r, c) \leftarrow \\ \quad \exists s\ \mathit{ContainsShipS}(r, c, s)) \end{array} \right\}, \tag{3.16}$$

$$\left\{ \begin{array}{l} \forall r_1 \forall c_1 \forall r_2 \forall c_2\ (\mathit{Adjacent}(r_1, c_1, r_2, c_2) \leftarrow \\ \quad \mathrm{abs}(r_1 - r_2) \leq 1 \wedge \mathrm{abs}(c_1 - c_2) \leq 1) \end{array} \right\}, \tag{3.17}$$

$$\begin{array}{l} \forall r_1 \forall c_1 \forall r_2 \forall c_2 \forall s_1 \forall s_2\ (\\ \quad \mathit{ContainsShipS}(r_1, c_1, s_1) \wedge \mathit{ContainsShipS}(r_2, c_2, s_2) \wedge s_1 \neq s_2 \\ \quad \Rightarrow \neg \mathit{Adjacent}(r_1, c_1, r_2, c_2)), \end{array} \tag{3.18}$$

$$\forall r\ \ (\textsc{card}\{c \mid \mathit{ContainsShip}(r, c)\} = \mathit{RowNumber}(r))\,, \tag{3.19}$$

$$\forall c\ \ (\textsc{card}\{r \mid \mathit{ContainsShip}(r, c)\} = \mathit{ColNumber}(c))\,. \tag{3.20}$$

Definition (3.15) defines which squares contain part of ship s in terms of the position of the first part of s, its direction and its length. Definition (3.16) defines *ContainsShip* in terms of *ContainsShipS*, definition (3.17) expresses what adjacent squares are. Next, (3.18) states that two squares containing different ships cannot be adjacent. The two sentences (3.19) and (3.20) were explained in Example 3.8.

3.8 Note on related languages

As we mentioned in the beginning of this chapter, FO(·) is a moving target. Several other extensions of FO can be thought of and added to FO(·). One of the extensions that is currently being investigated in our research group is FO(FD) (Hou and Denecker, 2009; Hou, 2010), an extension of FO with fixpoint definitions. Both least and greatest fixpoint definitions are allowed in FO(FD). Moreover, these definitions can be nested arbitrarily deep.

In the literature, many languages are proposed that are syntactic variants of FO and/or extend FO with different constructs. Often, these languages are designed with a particular inference method in mind. For example, description logics are (mostly) fragments of FO that allow for describing ontologies. Description logics are designed to make deduction at least decidable, and preferably tractable. Other logics were designed to be used with finite model generation as inference. To this category belong the input languages of Answer Set Programming (ASP) systems (Syrjänen, 2000; Gebser et al., 2009a; Leone et al., 2006), NP-SPEC (Cadoli et al., 2000), PSGRND (East and Truszczyński, 2006) and the ALLOY

analyzer (Jackson, 2006), and languages to formulate constraint satisfaction problems (e.g., Michel and Van Hentenryck, 2005). Still other logics aim at describing planning problems, dynamic systems, games, etc. It is beyond the scope of this thesis to discuss the relation of all these languages to FO(·). See the work of Mariën et al. (2004), East and Truszczyński (2006) and Mariën et al. (2006) for interesting relations between FO(ID) and ASP. Denecker and Ternovska (2007) describe the use of FO(ID) for describing temporal reasoning domains.

3.9 Conclusion

In this chapter, we presented an extension of classical logic with partial functions, subsorts, arithmetic, aggregates and inductive definitions. The resulting logic is called FO(·) and shares important properties with classical logic. First of all, it has a clear formal and informal semantics, which are necessary features of a logic underlying a knowledge base system. Secondly, its data-complexity remains in polynomial time. We will see in the next chapters that this implies there exists polynomial time algorithms for several useful forms of inference on FO(·) theories. FO(·) improves on FO in the sense that its extensions, in particular the aggregates and inductive definitions, often allow for a much more natural and concise representation of knowledge. We conclude that FO(·) is a good candidate for a logic of a knowledge base system.

Chapter 4

Constraint propagation

In this chapter, we investigate constraint propagation for FO($\cdot$). Constraint propagation is a basic form of inference that can be used in implementations of many other types of inference, as we will see in the next chapters.
We explain constraint propagation on an example. Consider a database application allowing university students to compose their curriculum by selecting certain didactic modules and courses. Assume that amongst others, the following integrity constraints are imposed on the selections:

$$\begin{aligned}
&\neg(\mathit{Selected}(C_1) \wedge \mathit{Selected}(C_2)),\\
&\exists m\ (\mathit{Module}(m) \wedge \mathit{Selected}(m)),\\
&\forall m\ ((\mathit{Module}(m) \wedge \mathit{Selected}(m)) \Rightarrow\\
&\qquad \forall c\ ((\mathit{Course}(c) \wedge \mathit{In}(m,c)) \Rightarrow \mathit{Selected}(c))).
\end{aligned}$$

The first constraint states that courses C_1 and C_2 are mutually exclusive, the second one expresses that at least one module should be taken and the third one ensures that all courses of a selected module are selected. Assume that at some point in the application, the student has selected some modules or courses, rejected some others and is still undecided about the rest. That is, an incomplete database (three-valued structure) is given. It is possible now to use the integrity constraints to derive more complete information about the final selection of the student. For example, if course C_1 has been selected, it can be derived that C_2 is not going to be in the selection. Then, also every module that contains C_2 cannot be selected. If it is derived that all models except a certain module m will not be in the selection, then m and all its courses will be in the selection.
A method to perform constraint propagation on a theory T is by representing the incomplete database as a finite three-valued structure $\tilde{I}$ and then computing the set $\mathcal{M}$ of all models of T that complete the database, i.e. $\mathcal{M} = \{M \mid M \models T \text{ and } \tilde{I} \leq_p M\}$. Everything that is true, respectively false, in all $M \in \mathcal{M}$ is then derived to be true, respectively false. We call the operator that implements this propagation the *complete propagator*. In general, applying the complete

propagator is too expensive to be used in realistic applications. In this chapter, we develop a less expensive constraint propagation method.
The method we propose consists of two steps. First, the theory T is rewritten to an equivalent theory T' in a normal form called *equivalence normal form*. For each of the constraints of T', the complete propagator has polynomial time data-complexity and can easily be implemented. By successively applying the complete propagators for the constraints in T', propagation with polynomial time data-complexity for T', and hence for T, is obtained.
Besides the lower complexity compared to applying the complete propagator, there are two other benefits of our method. First, the propagation can be represented by an inductive definition. Therefore, algorithms to compute the model of a definition can be used to make an efficient implementation of our method. Also, theoretical results about program analysis and transformations become available to investigate the propagation for a theory. Secondly, it is possible to execute the propagation symbolically, i.e., independent of the given three-valued structure. Symbolic propagation is useful in, e.g., approximate query answering (Cortés-Calabuig et al., 2008).
The rest of this chapter is organized as follows. First we formally introduce constraint propagation and complete propagators. In Section 4.2, we present our constraint propagation method for FO. The symbolic version of the method is presented in Section 4.3. Next, we extend the method to FO($\cdot$). Finally, several applications of constraint propagations are discussed: finite model generation, building configuration systems and approximate query answering for incomplete databases with integrity constraints. Two other applications, namely improving grounders and debugging logic theories are presented in, respectively, Chapter 5 and Chapter 6.

4.1 Propagation on logic theories

In this section, we introduce the basic concepts about constraint propagation. To facilitate the presentation, we work with one-sorted logic in the theoretical parts this chapter. The examples may use many-sorted logic.

4.1.1 Propagators

For the rest of this chapter, let T be a theory and $\tilde{I}$ a four-valued structure with *finite* domain D. The structure $\tilde{I}$ can be seen as approximating some models of T, namely all two-valued structures M such that $\tilde{I} \leq_p M$ and $M \models T$. The goal of constraint propagation for T is then to find a better approximation of these models, i.e., one that is more precise than $\tilde{I}$. We call an operator on the class of finite four-valued Σ-structures a *propagator for* T if it performs constraint propagation for T. Formally, O is a propagator for T if the following two conditions are met:

1. $\tilde{I} \leq_p O(\tilde{I})$ for every structure $\tilde{I}$.

2. If $J \models T$ and $\tilde{I} \leq_p J$, then also $O(\tilde{I}) \leq_p J$.

The first condition states that by applying an operator no information is lost. The second condition states that no models of T approximated by $\tilde{I}$ are lost. Note that for a propagator O it follows from the definition above that $\tilde{I}$ and $O(\tilde{I})$ must have the same domain.
A propagator O is called *monotone* if for every two structures $\tilde{I}$ and $\tilde{J}$ such that $\tilde{I} \leq_p \tilde{J}$, also $O(\tilde{I}) \leq_p O(\tilde{J})$ holds. An example of a monotone propagator is the *inconsistency propagator* INCO, defined by

$$\text{INCO}(\tilde{I}) = \begin{cases} \tilde{I} & \text{if } \tilde{I} \text{ is three-valued} \\ \top^{\leq_p} & \text{otherwise} \end{cases}$$

The composition of two propagators is a propagator itself.

Lemma 4.1. *If O_1 is a propagator for T_1 and O_2 a propagator for T_2, then $O_1 \circ O_2$ is a propagator for $T_1 \cup T_2$.*

Proof. Since O_1 and O_2 are propagators, $\tilde{I} \leq_p O_2(\tilde{I}) \leq_p O_1(O_2(\tilde{I})) = (O_1 \circ O_2)(\tilde{I})$ for every structure $\tilde{I}$. If $J \models T_1 \cup T_2$ and $\tilde{I} \leq_p J$, then $O_2(\tilde{I}) \leq_p J$ and therefore also $O_1(O_2(\tilde{I})) \leq_p J$. ■

4.1.2 Refinement sequences

If V is a set of propagators for T, Lemma 4.1 ensures constraint propagation for T can be performed by starting from $\tilde{I}$ and successively applying propagators from V. We then get a sequence of increasingly precise four-valued structures. If such a sequence is strictly increasing in precision, we call it a *V-refinement sequence from $\tilde{I}$*.

Definition 4.1. Let V be a set of propagators for T. We call a sequence $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$ of four-valued structures a *V-refinement sequence from $\tilde{I}$* if the following conditions are met:

- $\tilde{J}_0 = \tilde{I}$;
- $\tilde{J}_i <_p \tilde{J}_{i+1}$ for every $0 \leq i < n$;
- for every $0 \leq i < n$, there exists an $O \in V$ such that $\tilde{J}_{i+1} = O(\tilde{J}_i)$.

Since refinement sequences are strictly increasing in precision, it follows that every refinement sequence from a finite structure $\tilde{I}$ is finite. Moreover, the length of a refinement sequence from $\tilde{I}$ is polynomial in the size of $\tilde{I}$.

Proposition 4.2. *The length of a refinement sequence from a finite structure $\tilde{I}$ is polynomial in $|\tilde{I}|$.*

Proof. See Appendix A. ■

A refinement sequence is called *terminal* if it cannot be extended anymore. For a terminal refinement sequence $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$, the structure $\tilde{J}_n$ is called its *limit*. If V only contains monotone propagators, the limit of a terminal V-refinement sequence from a finite structure is unique.

Proposition 4.3. *Let V be a set of monotone propagators for T and let $\tilde{I}$ be a finite structure. Then every terminal V-refinement sequence from $\tilde{I}$ has the same limit.*

Proof. Let $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$ and $\langle \tilde{K}_j \rangle_{0 \leq j \leq m}$ be two terminal V-refinement sequences from $\tilde{I}$. Denote by $O_{\tilde{J}_i}$, respectively $O_{\tilde{K}_j}$, the propagators from V such that $O_{\tilde{J}_i}(\tilde{J}_i) = \tilde{J}_{i+1}$, respectively $O_{\tilde{K}_j}(\tilde{K}_j) = \tilde{K}_{j+1}$. Because each propagator in V is monotone, it follows that

$$\begin{aligned}
\tilde{K}_m &= O_{\tilde{K}_{m-1}}\left(O_{\tilde{K}_{m-2}}\left(\cdots\left(O_{\tilde{K}_0}\left(\tilde{I}\right)\right)\cdots\right)\right) \\
&\leq_p O_{\tilde{K}_{m-1}}\left(O_{\tilde{K}_{m-2}}\left(\cdots\left(O_{\tilde{K}_0}\left(\tilde{J}_n\right)\right)\cdots\right)\right) \\
&= \tilde{J}_n
\end{aligned}$$

Similarly, we can derive that $\tilde{J}_n \leq_p \tilde{K}_m$. Hence $\tilde{J}_n = \tilde{K}_m$. ■

If V only contains monotone propagators, we denote by $\lim_V$ the operator that maps every finite structure to the unique limit of any terminal V-refinement sequence from finite structure $\tilde{I}$. From Lemma 4.1 it follows that $\lim_V$ is a propagator.

4.1.3 Complete propagators

The *complete propagator* for a theory T is the propagator that yields the most precise structures. This propagator is denoted by $\mathscr{O}^T$ and defined by

$$\mathscr{O}^T(\tilde{I}) = \mathrm{glb}_{\leq_p}\left(\{M \mid \tilde{I} \leq_p M \text{ and } M \models T\}\right).$$

It is straightforward to check that $\mathscr{O}^T$ is indeed a propagator. It is also a monotone propagator.

Lemma 4.4. *For every theory T, $\mathscr{O}^T$ is a monotone propagator.*

Proof. Follows immediately from the fact that $\{M \mid \tilde{I} \leq_p M \text{ and } M \models T\}$ is a superset of $\{M \mid \tilde{J} \leq_p M \text{ and } M \models T\}$ if $\tilde{I} \leq_p \tilde{J}$. ■

Proposition 4.5. *Let O be a propagator for T and $\tilde{I}$ a structure. Then $O(\tilde{I}) \leq_p \mathscr{O}^T(\tilde{I})$.*

Proof. To prove the proposition, we show that $P^{O(\tilde{I})}(\overline{d}) \leq_p P^{\mathscr{O}^T(\tilde{I})}(\overline{d})$ for any domain atom $P(\overline{d})$. If $P^{O(\tilde{I})}(\overline{d}) = \mathbf{i}$, it follows from the fact that O is a propagator that there is no model of T approximated by $\tilde{I}$. From the definition of

$\mathscr{O}^T$, we conclude that also $P^{\mathscr{O}^T(\tilde{I})}(\overline{d}) = \mathbf{i}$. If on the other hand $P^{O(\tilde{I})}(\overline{d}) = \mathbf{t}$ or $P^{O(\tilde{I})}(\overline{d}) = \mathbf{f}$, then $P(\overline{d})$ is true, respectively false, in every model of T approximated by $\tilde{I}$. Therefore $P^{\mathscr{O}^T(\tilde{I})}(\overline{d}) \geq_p \mathbf{t}$, respectively $P^{\mathscr{O}^T(\tilde{I})}(\overline{d}) \geq_p \mathbf{f}$, in this case. It follows that $P^{O(\tilde{I})}(\overline{d}) \leq_p \mathscr{O}^T(\tilde{I})(\overline{d})$ for every domain atom of the form $P(\overline{d})$. The proof is similar for domain atoms of the form $F(\overline{d}) = d'$. ■

Proposition 4.6. *The complete propagator for a theory T is idempotent, i.e., $\mathscr{O}^T \circ \mathscr{O}^T = \mathscr{O}^T$.*

Proof. Follows immediately from Lemma 4.1 and Proposition 4.5. ■

Example 4.1. Let Σ be the vocabulary

$$\langle \{ CM, Course, Module \}, \{ Selected(CM), \\ In(Module, Course) \}, \{ C_1 : Course, C_2 : Course \} \rangle,$$

where *Course* and *Module* are subsorts of *CM*. Let $\tilde{I}_0$ be the Σ-structure that is two-valued on all symbols except *Selected* and is given by

$$Module^{\tilde{I}_0} = \{m_1, m_2\}, \qquad Course^{\tilde{I}_0} = \{c_1, c_2, c_3, c_4\},$$
$$CM^{\tilde{I}_0} = \{m_1, m_2, c_1, c_2, c_3, c_4\}, \qquad In^{\tilde{I}_0} = \{(m_1, c_1), (m_1, c_3), (m_2, c_2)\},$$
$$C_1^{\tilde{I}_0} = c_1, \qquad C_2^{\tilde{I}_0} = c_2,$$
$$Selected^{\tilde{I}_0^{\mathrm{ct}}} = \{c_1\}, \qquad Selected^{\tilde{I}_0^{\mathrm{cf}}} = \emptyset$$

This structure expresses that course c_1 is certainly selected, while it is unknown whether other modules or courses are selected. Let T_1 be the theory

$$\neg(Selected(C_1) \wedge Selected(C_2)) \tag{4.1}$$
$$\exists m \; Selected(m) \tag{4.2}$$
$$\forall c \; ((\exists m \; (Selected(m) \wedge In(m, c))) \Rightarrow Selected(c)), \tag{4.3}$$

where m and c are variables of sort *Module*, respectively *Course*. Then $\mathscr{O}^{T_1}(\tilde{I}_0)$ assigns

$$Selected^{(\mathscr{O}^{T_1}(\tilde{I}_1))^{\mathrm{ct}}} = \{m_1, c_1, c_3\},$$
$$Selected^{(\mathscr{O}^{T_1}(\tilde{I}_1))^{\mathrm{cf}}} = \{m_2, c_2\}.$$

Indeed, because c_1 is selected according to $\tilde{I}_0$, we can derive from (4.1) that c_2 cannot be selected. Next, (4.3) implies that module m_2 cannot be selected. It then follows from (4.2) that m_1 must be selected. This implies in turn that c_3 must be selected. No information about c_4 can be derived since both $\mathscr{O}^{T_1}(\tilde{I}_1)[Selected(c_4)/\mathbf{t}]$ and $\mathscr{O}^{T_1}(\tilde{I}_1)[Selected(c_4)/\mathbf{f}]$ are models of T_1.

Example 4.2. Let T is the theory of Example 3.20 (the theory expressing the constraints on the solution of a battleship puzzle) and $\tilde{I}$ the three-valued structure over T's vocabulary, describing the unsolved puzzle in Figure 2.1. One can check that this puzzle has only one solution, and therefore, this solution is described by the interpretation of *ContainsShip* in $\mathscr{O}^T(\tilde{I})$.

Observe that if T has no models approximated by $\tilde{I}$, then $\mathscr{O}^T(\tilde{I}) = \top^{\leq_p}$. In particular, this is the case if $\tilde{I}$ is strictly four-valued. Therefore, the problem of computing $\mathscr{O}^T(\tilde{I})$ is at least as hard as deciding whether T has a model approximated by $\tilde{I}$. If T is an FO theory, this problem is intractable: for a fixed T and varying $\tilde{I}$ it is **NP**-complete (Fagin, 1974).
Similarly as for theories, we associate to each sentence or definition φ the monotone propagator $\mathscr{O}^\varphi$, which maps a structure $\tilde{I}$ to the most precise approximation of φ from $\tilde{I}$. That is,

$$\mathscr{O}^\varphi(\tilde{I}) = \mathrm{glb}_{\leq_p}\left(\{M \mid \tilde{I} \leq_p M \text{ and } M \models \varphi\}\right).$$

Observe that for any sentence or definition $\varphi \in T$, $\mathscr{O}^T(\tilde{I})$ is more precise than $\mathscr{O}^\varphi(\tilde{I})$, since

$$\{J \mid \tilde{I} \leq_p J \text{ and } J \models T\} \subseteq \{J \mid \tilde{I} \leq_p J \text{ and } J \models \varphi\}$$

As such, we obtain the following lemma.

Lemma 4.7. *If φ is a sentence or definition of T, then $\mathscr{O}^\varphi$ is a monotone propagator for T.*

From Proposition 4.3 and Lemma 4.7 it follows that every terminal $\{\mathscr{O}^\varphi \mid \varphi \in T\}$-refinement sequence from finite structure $\tilde{I}$ has the same limit. We denote the propagator $\lim_{\{\mathscr{O}^\varphi \mid \varphi \in T\}}$ by $\mathscr{L}_T$. We call a $\{\mathscr{O}^\varphi \mid \varphi \in T\}$-refinement sequence also a *T-refinement sequence*.

Example 4.3 (Example 4.1 ctd.)**.** Let $\langle \tilde{I}_i \rangle_{0 \leq i \leq 4}$ be the T_1-refinement sequence from $\tilde{I}_0$ obtained by applying (in this order) the propagators $\mathscr{O}^{(4.1)}$, $\mathscr{O}^{(4.3)}$, $\mathscr{O}^{(4.2)}$ and $\mathscr{O}^{(4.3)}$. The same reasoning as we made in Example 4.1 shows that $c_2 \in \mathit{Selected}^{\tilde{J}_1^{\mathrm{cf}}}$, $m_2 \in \mathit{Selected}^{\tilde{J}_2^{\mathrm{cf}}}$, $m_1 \in \mathit{Selected}^{\tilde{J}_3^{\mathrm{ct}}}$ and $c_3 \in \mathit{Selected}^{\tilde{J}_4^{\mathrm{ct}}}$. Hence, $\tilde{J}_4 = \mathscr{O}^{T_1}(\tilde{I}_1)$, the refinement sequence is terminal and $\mathscr{L}_{T_1}(\tilde{I}_1) = \mathscr{O}^{T_1}(\tilde{I}_1)$.

Example 4.4. Let T be the theory of Example 3.20 and $\tilde{I}$ the three-valued structure that describes the unsolved battleship puzzle in Figure 2.1 (see Example 2.6). The situation described by $\mathscr{L}_T(\tilde{I})$ is shown in Figure 4.1. Part of the solution is detected in $\mathscr{L}_T(\tilde{I})$. For example, it is derived from sentence (3.20) and the fact that $\mathit{ColNumber}^{\tilde{I}}(1) = 0$ that the first column cannot contain ships. On the other hand, large parts of the solution are missing. For example, it is clear that $(3, 2)$ cannot contain a ship. However, deriving this information requires reasoning on (3.15), (3.16) and (3.18) simultaneously, while during the construction of $\mathscr{L}_T(\tilde{I})$, each of these definitions and sentences are propagated separately.

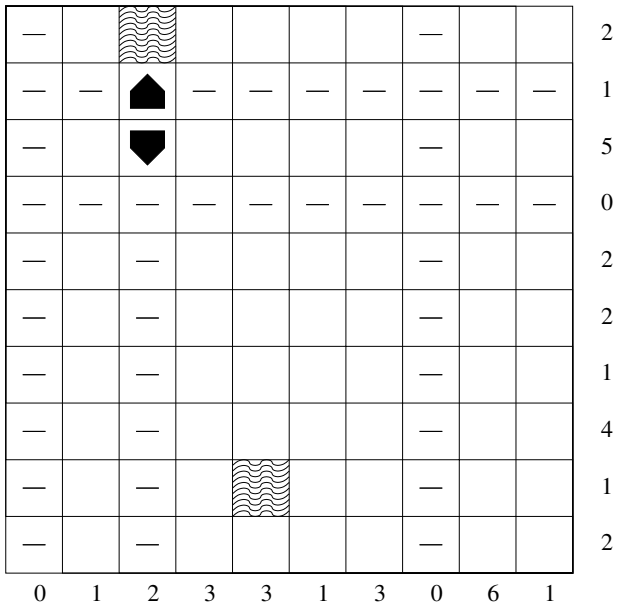


Figure 4.1: Constraint propagation on a battleship puzzle. The positions that cannot contain a ship according to $\mathscr{L}_T(\tilde{I})$ contain a '–'.

As Example 4.4 shows, it is not necessarily the case that $\mathscr{L}_T(\tilde{I}) = \mathscr{O}^T(\tilde{I})$. In general, only $\mathscr{L}_T(\tilde{I}) \leq_p \mathscr{O}^T(\tilde{I})$ holds. Note that $\mathscr{L}_T(\tilde{I}) = \mathscr{O}^T(\tilde{I})$ holds if T contains precisely one sentence or definition. Investigating other conditions that ensure $\mathscr{L}_T(\tilde{I}) = \mathscr{O}^T(\tilde{I})$ is part of future work.

4.2 Propagation for function-free FO

In this section, we introduce a constraint propagation method for function-free FO theories T that is computationally less expensive than applying $\mathscr{O}^T$ or computing a (terminal) T-refinement sequence. The method we propose is based on *implicational normal form propagators* (INF propagators). These propagators have several interesting properties. First, they are monotone, ensuring that terminal refinement sequences constructed using only INF propagators have a unique limit. Secondly, INF propagators have polynomial data-complexity and therefore terminal refinement sequences using only INF propagators can be computed in polynomial time. Thirdly, such a refinement sequence can be represented by a positive inductive definition, in the sense that the well-founded model of the definition extending a structure $\tilde{I}$ corresponds to the limit of a terminal refinement sequence from $\tilde{I}$. There are many systems available that can compute the well-founded model of a positive inductive definition and therefore can be applied to compute the limit of a terminal refinement sequence using

only INF propagators. Finally, INF propagators can be applied in a symbolic way, yielding propagation with constant data-complexity.
For the rest of this section, we assume that vocabularies contain no function symbols and all formulas are FO formulas. In Section 4.4 we consider propagation for full FO($\cdot$).

4.2.1 Implicational normal form propagators

INF propagators are associated to logic sentences in implicational normal form.

Definition 4.2. A sentence is in *implicational normal form (INF)* if it is of the form $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$, where ψ is an arbitrary formula and L is a literal.

For an INF sentence $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$, the associated INF propagator computes the value of ψ in the given structure. If this value is $\mathbf{t}$ or $\mathbf{i}$, the literal $L[\overline{x}]$ is made true, or inconsistent if it was false or inconsistent in the given structure. This is formalized in the following definition.

Definition 4.3. The operator $\mathscr{I}^{\varphi}$ associated to the sentence $\varphi := \forall \overline{x}\ (\psi \Rightarrow P(\overline{x}))$ is defined by

$$Q^{\mathscr{I}^{\varphi}(\tilde{I})}(\overline{d}_x) = \begin{cases} \mathrm{lub}_{\leq_p}\{\mathbf{t}, P^{\tilde{I}}(\overline{d}_x)\} & \text{if } Q = P \text{ and } \tilde{I}[\overline{x}/\overline{d}_x](\psi) \geq_p \mathbf{t} \\ Q^{\tilde{I}}(\overline{d}_x) & \text{otherwise} \end{cases}$$

The operator $\mathscr{I}^{\varphi}$ associated to the sentence $\varphi := \forall \overline{x}\ (\psi \Rightarrow \neg P(\overline{x}))$ is defined by

$$Q^{\mathscr{I}^{\varphi}(\tilde{I})}(\overline{d}_x) = \begin{cases} \mathrm{lub}_{\leq_p}\{\mathbf{f}, P^{\tilde{I}}(\overline{d}_x)\} & \text{if } Q = P \text{ and } \tilde{I}[\overline{x}/\overline{d}_x](\psi) \geq_p \mathbf{t} \\ Q^{\tilde{I}}(\overline{d}_x) & \text{otherwise} \end{cases}$$

Example 4.5. Sentence (4.3) is an INF sentence. Let $\tilde{I}$ be a structure such that $m_1 \in \mathit{Selected}^{\tilde{I}^{\mathrm{ct}}}$ and $(m_1, c_1) \in \mathit{In}^{\tilde{I}^{\mathrm{ct}}}$. Then $c_1 \in \mathit{Selected}^{(\mathscr{I}^{(4.3)}(\tilde{I}))^{\mathrm{ct}}}$. That is, if module m_1 is certainly selected and course c_1 certainly belongs to m_1, then the operator $\mathscr{I}^{(4.3)}$ associated to sentence (4.3) derives that c_1 is certainly selected. On the other hand, if $(m_2, c_2) \in \mathit{In}^{\tilde{I}^{\mathrm{ct}}}$ and $c_2 \in \mathit{Selected}^{\tilde{I}^{\mathrm{cf}}}$, the operator does not derive that m_2 is certainly not selected.

Proposition 4.8. *The operator $\mathscr{I}^{\varphi}$ is a monotone propagator for every INF sentence φ.*

Proof. Since φ is an INF sentence, it is of the form $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$. Denote $\mathrm{atom}(L[\overline{x}])$ by $P(\overline{x})$, i.e., $L[\overline{x}]$ is either the positive literal $P(\overline{x})$ or the negative literal $\neg P(\overline{z})$.
It follows directly from the definition of $\mathscr{I}^{\varphi}$ that $\tilde{I} \leq_p \mathscr{I}^{\varphi}(\tilde{I})$ for every structure $\tilde{I}$. Now let J be a structure such that $\tilde{I} \leq_p J$ and $J \models \varphi$. To show that $\mathscr{I}^{\varphi}$ is a propagator, we have to prove that $P^{\mathscr{I}^{\varphi}(\tilde{I})}(\overline{d}_x) \leq_p P^{J}(\overline{d}_x)$ for every tuple $\overline{d}_x$ of domain elements. If $\tilde{I}[\overline{x}/\overline{d}_x](\psi) \leq_p \mathbf{f}$ then $P^{\mathscr{I}^{\varphi}(\tilde{I})}(\overline{d}_x) = P^{\tilde{I}}(\overline{d}_x) \leq_p$

$P^J(\overline{d}_x)$. If on the other hand $\tilde{I}[\overline{x}/\overline{d}_x](\psi) = \mathbf{t}$, then also $J[\overline{x}/\overline{d}_x](\psi) = \mathbf{t}$ and therefore $J[\overline{x}/\overline{d}_x](L[\overline{x}]) = \mathbf{t}$. It follows that $\tilde{I}[\overline{x}/\overline{d}_x](L[\overline{x}]) \leq_p \mathbf{t}$ and hence $\mathscr{I}^{\varphi}(\tilde{I})[\overline{x}/\overline{d}_x](L[\overline{x}]) = \mathbf{t}$. We conclude that $P^{\mathscr{I}^{\varphi}(\tilde{I})}(\overline{d}_x) \leq_p P^J(\overline{d}_x)$.
The monotonicity of $\mathscr{I}^{\varphi}$ follows from the fact that $\tilde{I} \leq_p \tilde{J}$ implies $\tilde{I}\theta(\psi) \leq_p \tilde{J}\theta(\psi)$ for any two structures $\tilde{I}$ and $\tilde{J}$ and variable assignment θ. ■

As mentioned in Section 2.2.3, evaluating a formula in a four-valued structure $\tilde{I}$ takes polynomial time in $|\tilde{I}|$. It follows that computing $\mathscr{I}^{\varphi}(\tilde{I})$ takes polynomial time in $|\tilde{I}|$ for any INF sentence φ and structure $\tilde{I}$. If we combine this result with Proposition 4.2, we obtain the following theorem.

Theorem 4.9. *Let V be a finite set of INF sentences. Then $\lim_{\{\mathscr{I}^{\varphi} \mid \varphi \in V\}}(\tilde{I})$ is computable in polynomial time in $|\tilde{I}|$ for every finite structure $\tilde{I}$.*

Proof. Let $\varphi_1, \ldots, \varphi_n$ be all sentences in V. Let $\langle \tilde{J}_i \rangle_{0 \leq i \leq m}$ be longest sequence of structures such that $\tilde{J}_0 = \tilde{I}$ and $\tilde{J}_{i+1} = \mathscr{I}^{\varphi_k}(\tilde{J}_i)$, where k is the lowest number between 1 and n such that $\tilde{J}_i \neq \mathscr{I}^{\varphi_k}(\tilde{J}_i)$. Clearly, $\langle \tilde{J}_i \rangle_{0 \leq i \leq m}$ is a terminal $\{\mathscr{I}^{\varphi} \mid \varphi \in V\}$-refinement sequence from $\tilde{I}$. Proposition 4.2 implies that the length of this sequence is polynomial in $|\tilde{I}|$. Also, for each $i \geq 0$, $\tilde{J}_{i+1}$ can be computed in polynomial time in $|\tilde{I}|$: it suffices to compute $\mathscr{I}^{\varphi_1}(\tilde{J}_i)$, …, $\mathscr{I}^{\varphi_n}(\tilde{J}_i)$, and each $\mathscr{I}^{\varphi_k}(\tilde{J}_i)$, $1 \leq k \leq n$, can be computed in polynomial time in $|\tilde{J}_i| = |\tilde{I}|$. Hence $\langle \tilde{J}_i \rangle_{0 \leq i \leq m}$ can be computed in polynomial time in $|\tilde{I}|$. ■

4.2.2 Representing INF refinement sequences by an inductive definition

For the rest of this section, let V be a finite set of INF sentences and denote by $\mathscr{I}(V)$ the set $\{\mathscr{I}^{\varphi} \mid \varphi \in V\}$. We now show how to represent the propagator $\lim_{\mathscr{I}(V)}$ by an inductive definition Δ in the sense that for every structure $\tilde{I}$ $\lim_{\mathscr{I}(V)}(\tilde{I})$ is equal to the model of Δ extending $\tilde{I}$. One of the practical benefits of this representation is that any existing system to evaluate Δ can be applied to compute $\lim_{\mathscr{I}(V)}$.
Recall that Σ^{tf} is the vocabulary obtained from Σ by replacing every predicate symbol P/n by two predicate symbols P^{ct}/n and P^{cf}/n. Also, $\tilde{I}^{\mathrm{tf}}$ is the two-valued structure defined by $\overline{d} \in (P^{\mathrm{ct}})^{\tilde{I}^{\mathrm{tf}}}$ iff $P^{\tilde{I}}(\overline{d}) \geq_p \mathbf{t}$ and $\overline{d} \in (P^{\mathrm{cf}})^{\tilde{I}^{\mathrm{tf}}}$ iff $P^{\tilde{I}}(\overline{d}) \geq_p \mathbf{f}$. In Section 2.2.3 we showed how to construct for every Σ-formula φ a pair of negation-free formulas φ^{ct} and φ^{cf} such that for any variable assignment θ, $\tilde{I}^{\mathrm{tf}}\theta \models \varphi^{\mathrm{ct}}$ iff $\tilde{I}\theta(\varphi) \geq_p \mathbf{t}$ and $\tilde{I}\theta \models \varphi^{\mathrm{cf}}$ iff $\tilde{I}\theta(\varphi) \geq_p \mathbf{f}$ (see Proposition 2.7).
Denote by $\Sigma^{\mathrm{dom}(\tilde{I})}$ the vocabulary Σ extended with a new constant symbol $\mathbf{d}$ for every domain element d in the domain of $\tilde{I}$. We call these new constants *domain constants*. Abusing notation, we will denote both domain elements and their corresponding domain constants by d. If $\tilde{I}$ is a Σ-structure then we denote by $\tilde{I}$ also the expansion of $\tilde{I}$ to $\Sigma^{\mathrm{dom}(\tilde{I})}$ defined by assigning $\{d\}$ to every domain constant d.

Definition 4.4. Let V be a set of INF sentences and $\tilde{I}$ a structure. The *definition associated to* V is denoted by Δ_V and defined by

$$\Delta_V = \left\{ \forall \overline{x}\ ((L[\overline{x}])^{\mathrm{ct}} \leftarrow \psi^{\mathrm{ct}}) \mid \forall \overline{x}\ (\psi \Rightarrow L[\overline{x}]) \in V \right\}.$$

The *definition associated to* $\tilde{I}$ is denoted $\Delta_{\tilde{I}}$ and is defined by

$$\begin{aligned}\Delta_{\tilde{I}} = &\{P^{\mathrm{ct}}(\overline{d}) \leftarrow \top \mid P^{\tilde{I}}(\overline{d}) \geq_p \mathbf{t}\} \\ &\cup \{P^{\mathrm{cf}}(\overline{d}) \leftarrow \top \mid P^{\tilde{I}}(\overline{d}) \geq_p \mathbf{f}\} \\ &\cup \{P^{\mathrm{ct}}(\overline{d}) \leftarrow \bot \mid P^{\tilde{I}}(\overline{d}) \not\geq_p \mathbf{t}\} \\ &\cup \{P^{\mathrm{cf}}(\overline{d}) \leftarrow \bot \mid P^{\tilde{I}}(\overline{d}) \not\geq_p \mathbf{f}\}\end{aligned}$$

Definition $\Delta_{V,\tilde{I}}$ denotes the union of Δ_V and $\Delta_{\tilde{I}}$.

Example 4.6. Let $\tilde{I}_0$ be the structure given in Example 4.1 and let V be the following set of INF sentences[15]

$$Selected(C_1) \Rightarrow \neg Selected(C_2), \tag{4.4}$$
$$Selected(C_2) \Rightarrow \neg Selected(C_1), \tag{4.5}$$
$$\forall m\ ((\forall m'\ (m = m' \vee \neg Selected(m'))) \Rightarrow Selected(m)), \tag{4.6}$$
$$\forall c\ ((\exists m\ (Selected(m) \wedge In(m, c))) \Rightarrow Selected(c)), \tag{4.7}$$

where m and m' are variables of sort *Module* and c a variable of sort *Course*. The definition Δ_V is then given by

$$\left\{ \begin{array}{rl} Selected^{\mathrm{cf}}(C_2) & \leftarrow Selected^{\mathrm{ct}}(C_1) \\ Selected^{\mathrm{cf}}(C_1) & \leftarrow Selected^{\mathrm{ct}}(C_2) \\ \forall m\ (Selected^{\mathrm{ct}}(m) & \leftarrow \forall m'\ (m = m' \vee Selected^{\mathrm{cf}}(m'))) \\ \forall c\ (Selected^{\mathrm{ct}}(c) & \leftarrow \exists m\ (Selected^{\mathrm{ct}}(m) \wedge In^{\mathrm{ct}}(m, c))) \end{array} \right\}$$

The definition $\Delta_{\tilde{I}_0}$ is given by

$$\left\{ \begin{array}{rl} Selected^{\mathrm{ct}}(c_1) & \leftarrow \top \\ Selected^{\mathrm{ct}}(c_2) & \leftarrow \bot \\ Selected^{\mathrm{ct}}(m_1) & \leftarrow \bot \\ Selected^{\mathrm{ct}}(m_2) & \leftarrow \bot \\ In^{\mathrm{ct}}(m_1, c_1) & \leftarrow \top \\ In^{\mathrm{ct}}(m_1, c_3) & \leftarrow \top \\ \vdots & \end{array} \right\}$$

It follows directly from Proposition 2.8 that Δ_V, $\Delta_{\tilde{I}}$ and $\Delta_{V,\tilde{I}}$ are positive definitions, i.e., no defined predicate occurs negatively in the body of the rules in these definitions. Also note that $\Delta_{\tilde{I}}$ and $\Delta_{V,\tilde{I}}$ contain only defined symbols,

[15] These sentences can be derived automatically from the theory of Example 4.1. We show this in Section 4.2.3.

except for the domain constants. As a result $\text{wfm}_{\Delta_{\tilde{I}}}(\tilde{J}) = \text{wfm}_{\Delta_{\tilde{I}}}(\tilde{K})$ and $\text{wfm}_{\Delta_{V,\tilde{I}}}(\tilde{J}) = \text{wfm}_{\Delta_{V,\tilde{I}}}(\tilde{K})$ for any two structures $\tilde{J}$ and $\tilde{K}$ with the same domain. In other words, for a fixed domain, $\Delta_{\tilde{I}}$ and $\Delta_{V,\tilde{I}}$ have a unique model. The unique model of $\Delta_{\tilde{I}}$ and $\Delta_{V,\tilde{I}}$ with the same domain as $\tilde{I}$ corresponds to the *least Herbrand model* of $\Delta_{\tilde{I}}$, respectively $\Delta_{V,\tilde{I}}$.

Clearly, $\Delta_{\tilde{I}}$ can be seen as a description of $\tilde{I}$: the least Herbrand model of $\Delta_{\tilde{I}}$ coincides with $\tilde{I}^{\text{tf}}$. The following proposition states that the well-founded model of $\Delta_{V,\tilde{I}}$ coincides with the limit of a terminal refinement $\mathscr{I}(V)$-refinement sequence from $\tilde{I}$. Therefore, Δ_V can be seen as a description of $\lim_{\mathscr{I}(V)}$.

Proposition 4.10. *The least Herbrand model of* $\Delta_{V,\tilde{I}}$ *is equal to* $(\lim_{\mathscr{I}(V)}(\tilde{I}))^{\text{tf}}$.

Proof. See Appendix A. ■

There are several benefits of using Δ_V as a description of $\lim_{\mathscr{I}(V)}$. From a practical point of view, Proposition 4.10 states that we can use any existing algorithm that computes the least Herbrand model of positive definitions to implement $\lim_{\mathscr{I}(V)}$. Several such algorithms have been developed. For example, in the area of production rule systems, RETE (Forgy, 1982) and LEAPS (Miranker et al., 1990) are two well-known algorithms that accomplish this task. Other examples are the algorithms implemented in Prolog systems with tabling such as XSB (Swift, 2009) and YAP (Faustino da Silva and Santos Costa, 2006). In the context of databases, a frequently used algorithm is semi-naive evaluation (Ullman, 1988). Adaptions of the semi-naive evaluation are implemented in the grounding component of DLV (Perri et al., 2007) and in the grounder GIDL (see Chapter 5). It follows that the large amount of research on optimization techniques and execution strategies for these algorithms can be used to obtain efficient implementations of $\lim_{\mathscr{I}(V)}$ for a set V of INF propagators.

Most of the algorithms and systems mentioned above expect that all rules of a definition are of the form $\forall \overline{x}\ (P(\overline{x}) \leftarrow \exists \overline{y}\ (Q_1(\overline{z}_1) \wedge \ldots \wedge Q_n(\overline{z}_n)))$, i.e., each body is the existential quantification of a conjunction of atoms. Some of the algorithms, e.g., semi-naive evaluation, can easily be extended to more general definitions. Another solution consists of rewriting definitions into the desired format by applying predicate introduction for definitions (see Section 3.5.3). The only non-trivial step in this process is in rewriting rules of the form $\forall \overline{x}\ (P(\overline{x}) \leftarrow \forall \overline{y}\ \varphi))$. Vennekens et al. (2007) show how this step can be accomplished if the Herbrand domain of $\Delta_{V,\tilde{I}}$, i.e., the set of all terms over $\Sigma^{\text{dom}(\tilde{I})}$, is finite. Since we assumed in this section that vocabularies do not contain function symbols, the Herbrand domain of $\Delta_{V,\tilde{I}}$ is indeed finite.

Other benefits of representing $\lim_{\mathscr{I}(V)}$ by Δ_V stem from the area of logic program analysis. For instance, *abstract interpretation* (e.g., Bruynooghe, 1991) can be used to derive interesting properties of $\lim_{\mathscr{I}(V)}$, *program specialization* (Leuschel, 1997) to tailor Δ_V to a specific class of structures $\tilde{I}$, *folding* (Pettorossi and Proietti, 1998) to combine the application of several propagators, etc.

4.2.3 From FO to INF

As mentioned above, computing $\mathscr{O}^T(\tilde{I})$ can be computationally expensive. The same holds for $\mathscr{L}_T(\tilde{I})$. For instance, if T contains only one sentence, then $\mathscr{O}^T = \mathscr{L}_T$, and therefore the best known algorithms for applying $\mathscr{L}_T$ take exponential time in $|\tilde{I}|$. In this section, we present a computationally cheaper method for constraint propagation on function-free FO theories. The method consists of transforming, in linear time, a theory T into an equivalent set of INF sentences. Then propagation on T can be performed by applying the corresponding INF propagators. Theorem 4.9 ensures that this propagation has polynomial time data-complexity. The price for this improved efficiency is of course a loss in precision.

From FO to ENF

The transformation of theories to INF sentences works in two steps. First, a theory T is transformed into an equivalent set of sentences in *equivalence normal form* (ENF). Next, each ENF sentence is replaced by a set of INF sentences. We show that both steps can be executed in linear time. Also, we prove that no precision is lost in the second step, i.e., for each ENF sentence φ, there exists a set of INF sentences V such that $\mathscr{O}^{\varphi}(\tilde{I}) = \lim_{\mathscr{I}(V)}(\tilde{I})$.

Definition 4.5. An FO sentence φ is in *equivalence normal form (ENF)* if it is of the form[16] $\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$, such that

- P is a predicate symbol or the symbol $\top$;
- ψ is a formula of the form L, $(L_1 \wedge L_2)$, $(L_1 \vee L_2)$, $(\forall v\ L)$ or $(\exists v\ L)$, where L, L_1 and L_2 are literals;
- no predicate symbol occurs more than once in φ.

An FO theory is in ENF if all its sentences are.

We now show that every function-free FO theory T over a vocabulary Σ can be transformed into a Σ-equivalent ENF theory T'. The transformation is akin to the Tseitin transformation for propositional logic (Algorithm 2.1) and uses predicate introduction.

Algorithm 4.1. Given an FO theory T:

1. Move all negations in T inside until they are in the front of predicates (This can be done by applying the equivalences (2.9)–(2.12)).
2. Replace every sentence φ of T that is not of the form $\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$ by $\top \Leftrightarrow \varphi$.

[16] Recall that we denote by $\psi[\overline{x}]$ that $\overline{x}$ are precisely the free variables of ψ. Thus, Definition 4.5 implicitly states that in every ENF sentence $\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow \psi)$, the free variables of ψ are the free variables of $P(\overline{x})$.

3. As long as T in not in ENF:
   (a) Choose from T a sentence $\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$ that is not in ENF and let $\chi[\overline{y}]$ be either a direct subformula of ψ that is not a literal or an atomic subformula $Q(\overline{y})$ of $\psi[\overline{x}]$ such that Q occurs more than once in $P(\overline{x}) \Leftrightarrow \psi$).
   (b) Replace $\chi[\overline{y}]$ by $R(\overline{y})$ in ψ, where R is a new predicate.
   (c) Add the sentence $\forall\overline{y}\ (R(\overline{y}) \Leftrightarrow \chi[\overline{y}])$ to T.
4. Return T.

Clearly, the result of Algorithm 4.1 is an ENF theory. Since the number of subformulas in T is linear in the size of T and each subformula is replaced at most once by a predicate, the algorithm runs in linear time. The first step of the algorithm is not necessary, but yields smaller results.

Example 4.7 (Example 4.1 ctd.)**.** The result of applying Algorithm 4.1 on the theory T_1 from Example 4.1 is the theory

$$\begin{aligned}
&\top \Leftrightarrow \neg \mathit{Selected}(C_1) \vee \neg \mathit{Selected}(C_2)\\
&\top \Leftrightarrow \exists m\ \mathit{Selected}(m)\\
&\top \Leftrightarrow \forall c\ \mathit{Aux}_1(c)\\
&\forall c\ (\mathit{Aux}_1(c) \Leftrightarrow \mathit{Aux}_2(c) \vee \mathit{Selected}(c))\\
&\forall c\ (\mathit{Aux}_2(c) \Leftrightarrow \forall m\ \mathit{Aux}_3(m, c))\\
&\forall c \forall m\ (\mathit{Aux}_3(m, c) \Leftrightarrow \neg \mathit{Selected}(m) \vee \neg \mathit{In}(m, c))
\end{aligned}$$

Here, the predicates $\mathit{Aux}_1, \mathit{Aux}_2$ and Aux_3 are introduced by the algorithm.

Proposition 4.11. *Let T' be the result of applying Algorithm 4.1 on a theory T over Σ. Then T and T' are Σ-equivalent.*

Proof. Follows directly from Proposition 2.2. ■

Proposition 4.11 ensures propagators for T' can be used to implement propagators for T. Indeed, let T be a theory over Σ and T' an Σ-equivalent theory over $\Sigma' \supseteq \Sigma$. Let $\tilde{I}$ be a four-valued Σ-structure and denote by $\tilde{I}'$ the least precise expansion of $\tilde{I}$ to Σ'. For any propagator O for T', $\tilde{I}' \leq_p O(\tilde{I}')$ and hence $\tilde{I} = \tilde{I}'|_\Sigma \leq_p O(\tilde{I}')|_\Sigma$. If $J \models T$ and $\tilde{I} \leq_p J$, then there exists an expansion J' of J to Σ' such that $J' \models T'$. Therefore $O(\tilde{I}') \leq_p J'$ and $O(\tilde{I}')|_\Sigma \leq_p J$. It follows that expanding $\tilde{I}$ to $\tilde{I}'$, applying O and then restricting the result to Σ implements a propagator for T.

From ENF to INF

As shown in the previous section, every theory over Σ can be transformed into a Σ-equivalent ENF theory. Now we show that any ENF theory can be transformed into a logically equivalent theory only containing INF sentences. More

φ	INF(φ)
$\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow L[\overline{x}])$	$\forall \overline{x}\ (P(\overline{x}) \Rightarrow L[\overline{x}])$ $\forall \overline{x}\ (\neg L[\overline{x}] \Rightarrow \neg P(\overline{x}))$ $\forall \overline{x}\ (L[\overline{x}] \Rightarrow P(\overline{x}))$ $\forall \overline{x}\ (\neg P(\overline{x}) \Rightarrow \neg L[\overline{x}])$
$\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \forall y\ L[\overline{x}, y])$	$\forall \overline{x} \forall y\ (P(\overline{x}) \Rightarrow L[\overline{x}, y])$ $\forall \overline{x}\ ((\exists y\ \neg L[\overline{x}, y]) \Rightarrow \neg P(\overline{x}))$ $\forall \overline{x}\ ((\forall y\ L[\overline{x}, y]) \Rightarrow P(\overline{x}))$ $\forall \overline{x} \forall y\ (\neg P(\overline{x}) \wedge (\forall y'\ (y \neq y' \Rightarrow L[\overline{x}, y'])) \Rightarrow \neg L[\overline{x}, y])$
$\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \exists y\ L[\overline{x}, y])$	$\forall \overline{x} \forall y\ (P(\overline{x}) \wedge (\forall y'\ (y \neq y' \Rightarrow \neg L[\overline{x}, y'])) \Rightarrow L[\overline{x}, y])$ $\forall \overline{x}\ ((\forall y\ \neg L[\overline{x}, y]) \Rightarrow \neg P(\overline{x}))$ $\forall \overline{x}\ ((\exists y\ L[\overline{x}, y]) \Rightarrow P(\overline{x}))$ $\forall \overline{x} \forall y\ (\neg P(\overline{x}) \Rightarrow \neg L[\overline{x}, y])$
$\forall \overline{x} \forall \overline{y} \forall \overline{z}\ (P(\overline{x}, \overline{y}, \overline{z}) \Leftrightarrow L_1[\overline{x}, \overline{y}] \wedge L_2[\overline{x}, \overline{z}])$	$\forall \overline{x} \forall \overline{y}((\exists \overline{z}\ P(\overline{x}, \overline{y}, \overline{z})) \Rightarrow L_1[\overline{x}, \overline{y}])$ $\forall \overline{x} \forall \overline{y} \forall \overline{z}\ (\neg L_1[\overline{x}, \overline{y}] \Rightarrow \neg P(\overline{x}, \overline{y}, \overline{z}))$ $\forall \overline{x} \forall \overline{z}((\exists \overline{y}\ P(\overline{x}, \overline{y}, \overline{z})) \Rightarrow L_2[\overline{x}, \overline{z}])$ $\forall \overline{x} \forall \overline{y} \forall \overline{z}\ (\neg L_2[\overline{x}, \overline{z}] \Rightarrow \neg P(\overline{x}, \overline{y}, \overline{z}))$ $\forall \overline{x} \forall \overline{y} \forall \overline{z}\ (L_1[\overline{x}, \overline{y}] \wedge L_2[\overline{x}, \overline{z}] \Rightarrow P(\overline{x}, \overline{y}, \overline{z}))$ $\forall \overline{x} \forall \overline{y}((\exists \overline{z}\ (\neg P(\overline{x}, \overline{y}, \overline{z}) \wedge L_2[\overline{x}, \overline{z}])) \Rightarrow \neg L_1[\overline{x}, \overline{y}])$ $\forall \overline{x} \forall \overline{z}((\exists \overline{y}\ (\neg P(\overline{x}, \overline{y}, \overline{z}) \wedge L_1[\overline{x}, \overline{y}])) \Rightarrow \neg L_2[\overline{x}, \overline{z}])$
$\forall \overline{x} \forall \overline{y} \forall \overline{z}\ (P(\overline{x}, \overline{y}, \overline{z}) \Leftrightarrow L_1[\overline{x}, \overline{y}] \vee L_2[\overline{x}, \overline{z}])$	$\forall \overline{x} \forall \overline{y} \forall \overline{z}\ (\neg L_1[\overline{x}, \overline{y}] \wedge \neg L_2[\overline{x}, \overline{z}] \Rightarrow \neg P(\overline{x}, \overline{y}, \overline{z}))$ $\forall \overline{x} \forall \overline{y}((\exists \overline{z}\ (P(\overline{x}, \overline{y}, \overline{z}) \wedge \neg L_2[\overline{x}, \overline{z}])) \Rightarrow L_1[\overline{x}, \overline{y}])$ $\forall \overline{x} \forall \overline{z}((\exists \overline{y}\ (P(\overline{x}, \overline{y}, \overline{z}) \wedge \neg L_1[\overline{x}, \overline{y}])) \Rightarrow L_2[\overline{x}, \overline{z}])$ $\forall \overline{x} \forall \overline{y} \forall \overline{z}\ (L_1[\overline{x}, \overline{y}] \Rightarrow P(\overline{x}, \overline{y}, \overline{z}))$ $\forall \overline{x} \forall \overline{y}((\exists \overline{z}\ \neg P(\overline{x}, \overline{y}, \overline{z})) \Rightarrow \neg L_1[\overline{x}, \overline{y}])$ $\forall \overline{x} \forall \overline{y} \forall \overline{z}\ (L_2[\overline{x}, \overline{z}] \Rightarrow P(\overline{x}, \overline{y}, \overline{z}))$ $\forall \overline{x} \forall \overline{z}((\exists \overline{y}\ \neg P(\overline{x}, \overline{y}, \overline{z})) \Rightarrow \neg L_2[\overline{x}, \overline{z}])$

Table 4.1: From ENF to INF

precisely, for each ENF sentence φ there exists a set INF(φ) of INF sentences such that $\mathscr{O}^{\varphi}(\tilde{I}) = \lim_{\mathscr{I}(\mathrm{INF}(\varphi))}(\tilde{I})$ for every structure $\tilde{I}$. In other words, $\mathscr{O}^{\varphi}$ can be simulated by applying the INF propagators in $\mathscr{I}(\mathrm{INF}(\varphi))$. As a result, $\mathscr{O}^{\varphi}(\tilde{I})$ can be computed in polynomial time in $|\tilde{I}|$. Hence, if T is an ENF theory, then $\mathscr{L}_T(\tilde{I})$ can be computed in polynomial time in $|\tilde{I}|$. Combined with the results of the previous section, we obtain a propagation method for function-free FO that has polynomial data-complexity.

Table 4.1 defines for each possible ENF sentence φ the desired set INF(φ) of INF sentences. Roughly, for an ENF sentence φ of the form $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$, the set INF($\varphi$) is obtained as follows. First, φ is split into the two implications $\forall \overline{x}\ (P(\overline{x}) \Rightarrow \psi[\overline{x}])$ and $\forall \overline{x}\ (\psi[\overline{x}] \Rightarrow P(\overline{x}))$. If one of the implications is of the form $\forall \overline{x}\ ((\psi_1 \vee \psi_2) \Rightarrow P(\overline{x}))$ it is split in the two implications $\forall \overline{x}\ (\psi_1 \Rightarrow P(\overline{x}))$

and $\forall\overline{x}\ (\psi_2 \Rightarrow P(\overline{x}))$. Similarly, if it is of the form $\forall\overline{x}\ (P(\overline{x}) \Rightarrow (\psi_1 \wedge \psi_2))$, it is split in $\forall\overline{x}\ (P(\overline{x}) \Rightarrow \psi_1)$ and $\forall\overline{x}\ (P(\overline{x}) \Rightarrow \psi_2)$. Finally, the set $\mathrm{INF}(\varphi)$ contains for each atom $Q(\overline{y})$ that occurs positively, respectively negatively, in one of the obtained implications ξ, an INF sentence of the form $\forall\overline{y}\ (\chi \Rightarrow Q(\overline{y}))$, respectively $\forall\overline{y}\ (\chi \Rightarrow \neg Q(\overline{y}))$, that is equivalent to ξ. Intuitively, this ensures the presence of an INF sentence in $\mathrm{INF}(\varphi)$ that may derive $Q(\overline{y})$ is certainly true, respectively certainly false. As an example, consider the ENF sentence $\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow \exists y\ Q(\overline{x}, y))$. This sentence is split into

$$\forall\overline{x}\ (\exists y\ Q(\overline{x}, y) \Rightarrow P(\overline{x})) \tag{4.8}$$

$$\forall\overline{x}\ (P(\overline{x}) \Rightarrow \exists y\ Q(\overline{x}, y)) \tag{4.9}$$

The two INF sentences derived from (4.8) are (4.8) itself and the sentence $\forall\overline{x}\forall y\ (\neg P(\overline{x}) \Rightarrow \neg Q(\overline{x}, y))$. Note that both are equivalent to (4.8). The two INF sentences derived from (4.9) are $\forall\overline{x}\ ((\forall y\ \neg Q(\overline{x}, y)) \Rightarrow \neg P(\overline{x}))$ and $\forall\overline{x}\forall y\ ((P(\overline{x}) \wedge \forall y'\ (y \neq y' \Rightarrow \neg Q(\overline{x}, y'))) \Rightarrow Q(\overline{x}, y))$. The latter sentences expresses that $Q(\overline{x}, y)$ is true if $P(\overline{x})$ is true and $Q(\overline{x}, y')$ is false for every $y' \neq y$. This is equivalent to (4.9).
The following theorem states that $\mathscr{O}^{\varphi}$ can be computed using the propagators in $\mathscr{I}(\mathrm{INF}(\varphi))$.

Theorem 4.12. *$\mathscr{O}^{\varphi}(\tilde{I}) = \textsc{inco}(\lim_{\mathscr{I}(\mathrm{INF}(\varphi))}(\tilde{I}))$ for every ENF sentence φ and structure $\tilde{I}$.*

The inconsistency propagator INCO is needed in the theorem for a small technical problem. The only strictly four-valued structure that can be obtained by applying a complete propagator is the most precise structure $\top^{\leq_p}$. This is in general not the case for the propagator $\lim_{\mathscr{I}(\mathrm{INF}(\varphi))}$. Applying the inconsistency propagator solves this technical detail.

Proof. See Appendix A. ■

The following corollary follows directly from Theorem 4.12 and Theorem 4.9.

Corollary 4.13. *For every ENF sentence φ, $\mathscr{O}^{\varphi}(\tilde{I})$ is computable in polynomial time in $|\tilde{I}|$. For every ENF theory T, $\mathscr{L}_T(\tilde{I})$ is computable in polynomial time in $|\tilde{I}|$.*

4.2.4 Summary

Combining the results above yields a propagation method for function-free FO theories. Specifically, we obtain the following algorithm.

Algorithm 4.2. For an input theory T over Σ and a Σ-structure $\tilde{I}$:

1. Transform T to an ENF theory T' using Algorithm 4.1.
2. Transform T' to a set V of INF sentences using Table 4.1.

3. Construct a (terminal) $\mathscr{I}(V)$-refinement sequence from $\tilde{I}$. Denote the last element by $\tilde{J}$.

4. Return $(\text{INCO}(\tilde{J}))|_\Sigma$.

Note that this is an any-time algorithm: the refinement sequence constructed in the third step can be terminal, but this is not necessary. In either case, the algorithm implements a propagator for T. Since only INF propagators are used, the third step can be executed by representing $\lim_{\mathscr{I}(V)}$ as a definition and computing the model of that definition (see Section 4.2.2). In the following, we call Algorithm 4.2 the *propagation algorithm*. We denote by $\text{INF}(T)$ the set of INF sentences obtained by rewriting T to a set of INF sentences as in Algorithm 4.2.
As a summary, we apply the propagation algorithm on the theory and structure from Example 4.1

Example 4.8. Let T_1 and $\tilde{I}_0$ be the theory and structure from Example 4.1. Transforming T_1 to ENF produces the theory shown in Example 4.7. According to Table 4.1, the set of INF sentences associated to this theory contains, amongst others, the sentences

$$\mathit{Selected}(C_1) \Rightarrow \neg \mathit{Selected}(C_2) \tag{4.10}$$
$$\forall c\ (\top \Rightarrow \mathit{Aux}_1(c)) \tag{4.11}$$
$$\forall c\ (\mathit{Aux}_1(c) \wedge \neg \mathit{Selected}(c) \Rightarrow \mathit{Aux}_2(c)) \tag{4.12}$$
$$\forall c \forall m\ (\mathit{Aux}_2(c) \Rightarrow \mathit{Aux}_3(m, c)) \tag{4.13}$$
$$\forall m\ (\exists c\ (\mathit{Aux}_3(m, c) \wedge \mathit{In}(m, c)) \Rightarrow \neg \mathit{Selected}(m)) \tag{4.14}$$
$$\forall m\ (\forall m'\ (m \neq m' \Rightarrow \neg \mathit{Selected}(m')) \Rightarrow \mathit{Selected}(m)) \tag{4.15}$$
$$\forall m \forall c\ (\mathit{Selected}(m) \wedge \mathit{In}(m, c) \Rightarrow \neg \mathit{Aux}_3(m, c)) \tag{4.16}$$
$$\forall c\ (\exists m\ \neg \mathit{Aux}_3(m, c) \Rightarrow \neg \mathit{Aux}_2(c)) \tag{4.17}$$
$$\forall c\ (\mathit{Aux}_1(c) \wedge \neg \mathit{Aux}_2(c) \Rightarrow \mathit{Selected}(c)) \tag{4.18}$$

If one applies the associated INF propagators on $\tilde{I}_0$ in the order of the sentences above, the following information is derived. First, $\mathscr{I}^{(4.10)}$ derives that c_2 is certainly not selected. Next, $\mathscr{I}^{(4.11)}$ derives that $\mathit{Aux}_1(c)$ is certainly true for all courses c. $\mathscr{I}^{(4.12)}$ combines the derived information and concludes that $\mathit{Aux}_2(c_2)$ is certainly true. This in turn implies, by $\mathscr{I}^{(4.13)}$, that $\mathit{Aux}_3(m, c)$ is certainly true for, a.o., $m = m_2$ and $c = c_2$. $\mathscr{I}^{(4.14)}$ derives from the fact that c_2 belongs to m_2, that m_2 cannot be selected. Next, it is derived that m_1 is certainly selected by applying $\mathscr{I}^{(4.15)}$, and finally, applying $\mathscr{I}^{(4.18)} \circ \mathscr{I}^{(4.17)} \circ \mathscr{I}^{(4.16)}$ yields that c_3 is certainly selected. As such, exactly the same information as in $\mathscr{O}^{T_1}(\tilde{I}_0)$ is derived.

The following example gives another illustration of what the propagation algorithm can achieve.

Example 4.9. Consider the theory T_2, taken from some planning domain, consisting of the sentence

$$\forall a \forall a_p \forall t \ (Prec(a_p, a) \wedge Do(a, t) \Rightarrow \exists t_p \ (t_p < t \wedge Do(a_p, t_p))).$$

This sentence describes that some action a with precondition a_p can only be performed at timepoint t if a_p is performed at some earlier timepoint t_p. Let $\tilde{I}_2$ be a structure such that

$$\tilde{I}_2(Prec(d_0, d_1) \wedge \ldots \wedge Prec(d_{n-1}, d_n)) = \mathbf{t}.$$

$\tilde{I}_2$ indicates that there is a chain of n actions that need to be performed before d_n. The propagation algorithm can derive for input T_2 and $\tilde{I}_2$ that d_n can certainly not be performed before the $(n+1)$th timepoint.

A note on precision

Because of Proposition 4.5, the result $\tilde{J}$ of applying Algorithm 4.2 on input theory T and structure $\tilde{I}$ is less precise than $\mathscr{O}^T(\tilde{I})$. As we will show in, e.g., Example 4.10 and Example 4.19, there are cases where $\tilde{J}$ is strictly less precise than $\mathscr{O}^T(\tilde{I})$. It is an important question for which T and $\tilde{I}$ this loss in precision occurs. Cortés Calabuig (2008) presents some answers to this question in the context of incomplete databases under local closed world assumptions. It is a topic for future research to extend these results to more general contexts.
Because of Theorem 4.12 the loss in precision is due to the translation from a general FO theory to an ENF theory. It is noteworthy that a smarter algorithm than Algorithm 4.1 may lead to more precise propagation. We illustrate this on an example.

Example 4.10. Consider the propositional theory T consisting of the two sentences $(P \vee Q)$ and $((P \vee Q) \Rightarrow R)$. Clearly, R is true in every model of T and therefore $\mathscr{O}^T(\bot^{\leq_p})(R) = \mathbf{t}$. Algorithm 4.1 translates T to the ENF theory T' containing the sentences

$$\begin{aligned} &\top \Leftrightarrow P \vee Q, \\ &\top \Leftrightarrow Aux \vee R, \\ &Aux \Leftrightarrow \neg Q \wedge \neg P. \end{aligned}$$

One can check that $\mathscr{L}_{T'}(\bot^{\leq_p})(R) = \mathbf{u}$. Intuitively, this loss in precision is due to the fact that a three-valued structure cannot "store" the information that $(P \vee Q)$ is true in every model of T' if neither P nor Q is true in every model. A smarter translation to ENF could recognize that the subformula $(P \vee Q)$ occurs multiple times in T and substitutes all of this occurrences by the same auxiliary predicate, leading to the ENF theory T'' given by

$$\begin{aligned} &\top \Leftrightarrow \neg Aux, \\ &\top \Leftrightarrow Aux \vee R, \\ &Aux \Leftrightarrow \neg Q \wedge \neg P. \end{aligned}$$

In this case there is no loss in precision: $\mathscr{L}_{T''}(\bot^{\leq_p})(R) = \mathbf{t}$. The fact that $(P \vee Q)$ must be true is "stored" in the interpretation of Aux.

4.3 Symbolic propagation

In this section, we discuss the symbolic version of INF propagators. To this end, we first introduce the notion of a *symbolic structure*. Intuitively, a symbolic structure Φ over a vocabulary Σ assigns to every predicate P a query P^Φ. Given a database (structure) E, Φ represents a non-symbolic Σ-structure, namely the structure that assigns to P the set of answers to P^Φ in E. As such, Φ can be seen as a description of a set of structures, one for each possible database E to evaluate the queries.
Once symbolic structures are defined, symbolic INF propagators are introduced. These propagators map symbolic structures to symbolic structures, in a similar way as non-symbolic INF propagators do for non-symbolic structures. Applying a symbolic INF propagator associated to a sentence φ on a symbolic structure Φ therefore simulates applying $\mathscr{I}^\varphi$ at once to a whole class of structures, namely the class of structures described by Φ. Another benefit is that applying a symbolic INF propagator is much more efficient than applying the corresponding non-symbolic propagator. On the other hand, we will show that interesting properties of refinement sequences such as finiteness do not hold for sequences constructed using symbolic propagators.

4.3.1 Symbolic structures

For the rest of this section, let σ be a vocabulary, not necessarily related to Σ. This vocabulary will be used to describe structures in a symbolic way. We still assume that Σ does not contain function symbols.

Definition 4.6. A *symbolic two-valued Σ-structure* Φ over σ is an assignment of a query $\{\overline{x} \mid \varphi[\overline{y}]\}$ to every predicate $P/n \in \Sigma_{\text{pred}}$, where $|\overline{x}| = n$, $\overline{y} \subseteq \overline{x}$ and φ is a formula over σ.

We denote by P^Φ the query assigned by symbolic structure Φ to predicate symbol P.
Given a fixed σ-structure E with domain D, we can evaluate a symbolic Σ-structure over σ and obtain a two-valued Σ-structure $E(\Phi)$ with domain D.

Definition 4.7. The *value of a symbolic Σ-structure Φ over σ in σ-structure* E is the Σ-structure $E(\Phi)$ defined by $P^{E(\Phi)} = (P^\Phi)^E$.

Given E, Φ can be used as a symbolic description of $E(\Phi)$. Given a set V of σ-structures, $\tilde{\Phi}$ can be seen as describing the set $\{E(\tilde{\Phi}) \mid E \in V\}$ of Σ-structures.

Example 4.11. Let Σ be the vocabulary from Example 4.1 and let σ be the vocabulary that contains every symbol of Σ, except the predicate *Selected*. Let $\tilde{I}_0$ be the Σ-structure from Example 4.1. Note that $\tilde{I}_0|_\sigma$ is a two-valued structure. Denote this structure by E. Define the symbolic $\mathscr{G}(\Sigma)$-structure Φ over σ

by $In^\Phi = \{(m,c) \mid In(m,c)\}$, $Selected^\Phi = \{c \mid c = C_1\}$, $\mathscr{G}^\Phi_{C_1} = \{c \mid c = C_1\}$ and $\mathscr{G}^\Phi_{C_2} = \{c \mid c = C_2\}$. Evaluating Φ in E yields a $\mathscr{G}(\Sigma)$-structure that corresponds to $\tilde{I}^{\mathrm{ct}}_0$. For example, $Selected^{E(\Phi)} = (Selected^\Phi)^E = \{c \mid c = C_1\}^E = \{c_1\} = Selected^{\tilde{I}^{\mathrm{ct}}_0}$.

For a formula φ over Σ, its value $\Phi(\varphi)$ in symbolic structure Φ over σ is the formula obtained by replacing each occurrence of an atom $P(\overline{y})$ in φ by $\psi[\overline{x}/\overline{y}]$, where $P^\Phi = \{\overline{x} \mid \psi\}$. As such, $\Phi(\varphi)$ is a formula over σ. The following lemma states that $\Phi(\varphi)$ is a description of the truth value of φ in a structure described by Φ.

Lemma 4.14. *For every formula φ over Σ, symbolic Σ-structure Φ over σ, σ-structure E and variable assignment θ, $(E(\Phi))\theta \models \varphi$ iff $E\theta \models (\Phi(\varphi))$.*

Proof. If φ is the atomic formula $P(\overline{y})$ and $P^\Phi = \{\overline{x} \mid \psi\}$, then $E(\Phi)\theta \models P(\overline{y})$ iff $\theta(\overline{y}) \in P^{E(\Phi)}$ iff $\theta(\overline{y}) \in \{\overline{x} \mid \psi\}^E$ iff $E\theta[\overline{x}/\theta(\overline{y})] \models \psi$ iff $E\theta \models \psi[\overline{x}/\overline{y}]$ iff $E\theta \models \Phi(P(\overline{y}))$. The cases were φ is not atomic easily follow by induction. ■

Example 4.12 (Example 4.11 ctd.)**.** Let φ be the sentence

$$\forall c\ ((\exists m\ (Selected(m) \wedge In(m,c))) \Rightarrow Selected(c)). \tag{4.19}$$

Then φ^Φ is the sentence

$$\forall c\ ((\exists m\ (m = C_1 \wedge In(m,c))) \Rightarrow c = C_1).$$

Clearly, this sentence is satisfied in E, since $m = C_1$ is not satisfied for any module m. According to Lemma 4.14, therefore also $E(\Phi) \models \varphi$. Indeed, $Selected(m)$ is not true in $E(\Phi)$ for any module m since $Selected^{E(\Phi)} = \{c_1\}$, and hence φ is satisfied in $E(\Phi)$.

A *four-valued symbolic structure* $\tilde{\Phi}$ over σ is a pair $(\tilde{\Phi}^{\mathrm{ct}}, \tilde{\Phi}^{\mathrm{cf}})$ of two-valued symbolic structures over σ. As for non-symbolic structures, we denote the pair $(P^{\tilde{\Phi}^{\mathrm{ct}}}, P^{\tilde{\Phi}^{\mathrm{cf}}})$ by $P^{\tilde{\Phi}}$. The value of $\tilde{\Phi}$ in σ-structure E is the four-valued Σ-structure $\tilde{I}$ defined by $(\tilde{I}^{\mathrm{ct}}, \tilde{I}^{\mathrm{cf}}) = (E(\tilde{\Phi}^{\mathrm{ct}}), E(\tilde{\Phi}^{\mathrm{cf}}))$. Hence a four-valued symbolic structure can be seen as describing a class of four-valued non-symbolic structures.[17] The two-valued symbolic Σ^{tf}-structure $\tilde{\Phi}^{\mathrm{tf}}$ over σ is defined by $(P^{\mathrm{ct}})^{\tilde{\Phi}^{\mathrm{tf}}} = P^{\tilde{\Phi}^{\mathrm{ct}}}$ and $(P^{\mathrm{cf}})^{\tilde{\Phi}^{\mathrm{tf}}} = P^{\tilde{\Phi}^{\mathrm{cf}}}$. It is straightforward to check that $E(\tilde{\Phi}^{\mathrm{tf}}) = E(\tilde{\Phi})^{\mathrm{tf}}$ for every σ-structure E.
The value $\tilde{\Phi}(\varphi)$ of a formula φ in a symbolic structure $\tilde{\Phi}$ is the pair of σ-formulas $(\tilde{\Phi}^{\mathrm{tf}}(\varphi^{\mathrm{ct}}), \tilde{\Phi}^{\mathrm{tf}}(\varphi^{\mathrm{cf}}))$. Combining Lemma 4.14 and Proposition 2.7 then yields the desired result that the $\tilde{\Phi}(\varphi)$ is a description of the truth value of φ in a structure described by $\tilde{\Phi}$.

[17] Another interesting viewpoint is to see a symbolic Σ-structure $\tilde{\Phi}$ over σ as a description of the class of σ-structures E such that $E(\tilde{\Phi})$ contains no inconsistencies, i.e., is three-valued. Applying propagating on a symbolic structure $\tilde{\Phi}$ (see below) reduces this class, i.e., it enlarges the set of constraints a structure E must satisfy such that $E(\tilde{\Phi})$ is three-valued. Applications of this method to generate constraints on E are currently being investigated.

Lemma 4.15. *Let φ be a formula over Σ, $\tilde{\Phi}$ a four-valued Σ-structure over σ, E a σ-structure and θ a variable assignment. Then $E(\tilde{\Phi})\theta(\varphi) = E\theta(\tilde{\Phi}(\varphi))$.*

Proof. The first of the following equalities follows from Proposition 2.7, the second from the fact that $(E(\tilde{\Phi}))^{\mathrm{tf}} = E(\tilde{\Phi}^{\mathrm{tf}})$, the third one from Lemma 4.14 and the last one from the definition of $\tilde{\Phi}(\varphi)$.

$$\begin{aligned} E(\tilde{\Phi})\theta(\varphi) &= (E(\tilde{\Phi})^{\mathrm{tf}}\theta(\varphi^{\mathrm{ct}}), E(\tilde{\Phi})^{\mathrm{tf}}\theta(\varphi^{\mathrm{cf}})) \\ &= (E(\tilde{\Phi}^{\mathrm{tf}})\theta(\varphi^{\mathrm{ct}}), E(\tilde{\Phi}^{\mathrm{tf}})\theta(\varphi^{\mathrm{cf}})) \\ &= (E\theta(\tilde{\Phi}^{\mathrm{tf}}(\varphi^{\mathrm{ct}})), E\theta(\tilde{\Phi}^{\mathrm{tf}}(\varphi^{\mathrm{cf}}))) \\ &= E\theta(\tilde{\Phi}(\varphi)) \end{aligned}$$

■

Example 4.13 (Example 4.11 ctd.)**.** Let $\tilde{\Phi}^{\mathrm{ct}}$ be the symbolic structure Φ of Example 4.13. As explained that example, $E(\tilde{\Phi}^{\mathrm{ct}})$ is precisely the structure $\tilde{I}_0^{\mathrm{ct}}$. Let $\tilde{\Phi}^{\mathrm{cf}}$ be the symbolic $\mathscr{G}(\Sigma)$-structure over σ defined by $In^{\tilde{\Phi}^{\mathrm{cf}}} = \{(m,c) \mid \neg In(m,c)\}$, $Selected^{\tilde{\Phi}^{\mathrm{cf}}} = \{x \mid \bot\}$, $\mathscr{G}_{C_1}^{\tilde{\Phi}^{\mathrm{cf}}} = \{c \mid c \neq C_1\}$ and $\mathscr{G}_{C_2}^{\tilde{\Phi}^{\mathrm{cf}}} = \{c \mid c \neq C_2\}$. It can easily be checked that $E(\tilde{\Phi}^{\mathrm{cf}})$ corresponds to $\tilde{I}_0^{\mathrm{cf}}$. Let $\tilde{\Phi}$ be the four-valued $\mathscr{G}(\Sigma)$-structure $(\tilde{\Phi}^{\mathrm{ct}}, \tilde{\Phi}^{\mathrm{cf}})$. Then $E(\tilde{\Phi})$ corresponds to $\tilde{I}_0$.
Let φ be INF sentence (4.19) from example 4.12. Then φ^{ct} and φ^{cf} are given by, respectively,

$$\begin{aligned} &\forall c\ ((\exists m\ (\neg Selected^{\mathrm{cf}}(m) \wedge \neg In^{\mathrm{cf}}(m,c))) \Rightarrow Selected^{\mathrm{ct}}(c)), \\ &\forall c\ ((\exists m\ (\neg Selected^{\mathrm{ct}}(m) \wedge \neg In^{\mathrm{ct}}(m,c))) \Rightarrow Selected^{\mathrm{cf}}(c)). \end{aligned}$$

The evaluation of φ^{ct} and φ^{cf} in $\tilde{\Phi}^{\mathrm{tf}}$ are, respectively, the sentences

$$\begin{aligned} &\forall c\ ((\exists m\ (\neg\bot \wedge \neg\neg In(m,c))) \Rightarrow c = C_1), \\ &\forall c\ ((\exists m\ (\neg c = C_1 \wedge \neg In(m,c))) \Rightarrow \bot), \end{aligned}$$

which are simplified to $\forall c\ ((\exists m\ In(m,c)) \Rightarrow c = C_1)$, respectively $\forall c \forall m\ (c = C_1 \vee In(m,c))$. Both sentences are false in E, and therefore φ is unknown in $E(\tilde{\Phi})$.

4.3.2 Symbolic propagators

We now lift propagators to the symbolic level.

Definition 4.8. A *symbolic propagator* S for a theory T is an operator on the set of symbolic structures over σ such that for each σ-structure E and symbolic structure $\tilde{\Phi}$, the following conditions are satisfied:

- $E(\tilde{\Phi}) \leq_p E(S(\tilde{\Phi}))$;
- if $J \models T$ and $J \geq_p E(\tilde{\Phi})$, then also $J \geq_p E(S(\tilde{\Phi}))$.

Note that these two conditions on symbolic propagators are similar to the conditions on non-symbolic propagators. As is the case for non-symbolic propagators, the composition $S_2 \circ S_1$ of two symbolic propagators for theory T is again a symbolic propagator for T.
We say that a symbolic propagator S *describes* a non-symbolic propagator O if for every σ-structure E, $E \circ S = O \circ E$. It is straightforward to check that if S_1 describes O_1 and S_2 describes O_2, then $S_2 \circ S_1$ describes $O_2 \circ O_1$. It follows that symbolic propagators can be used to compute refinement sequences. Indeed, let V be a set of propagators such that for each $O \in V$, there exists a symbolic propagator S describing O. Let $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$ be a V-refinement sequence from $E(\tilde{\Phi})$ and denote by O_i a propagator such that $O_i(\tilde{J}_i) = \tilde{J}_{i+1}$. Then $\tilde{J}_n = E((S_{n-1} \circ \cdots \circ S_0)(\tilde{\Phi}))$, where S_i denotes the symbolic propagator that describes O_i for $0 \leq i < n$. As such, $(S_{n-1} \circ \cdots \circ S_0)(\tilde{\Phi})$ can be seen as a symbolic representation of the refinement sequence $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$.
We now introduce symbolic INF propagators. For the rest of this section, let $\tilde{\Phi}$ be a four-valued symbolic Σ-structure over σ and E a σ-structure. If two queries $\{\overline{x} \mid \psi\}$ and $\{\overline{y} \mid \chi\}$ have the same arity, i.e., $|\overline{x}| = |\overline{y}|$, we denote by $\{\overline{x} \mid \psi\} \cup \{\overline{y} \mid \chi\}$ the query $\{\overline{z} \mid \psi[\overline{x}/\overline{z}] \vee \chi[\overline{y}/\overline{z}]\}$, where $\overline{z}$ is a tuple of new variables. Note that $(\{\overline{x} \mid \psi\} \cup \{\overline{y} \mid \chi\})^E = \{\overline{x} \mid \psi\}^E \cup \{\overline{y} \mid \chi\}^E$ for every structure E.

Definition 4.9. Let φ be the INF sentence $\forall \overline{x}\ (\psi \Rightarrow P(\overline{x}))$. The *symbolic INF propagator* $\mathscr{I}_s^{\varphi}$ is defined by

$$Q^{\mathscr{I}_s^{\varphi}(\tilde{\Phi})} = \begin{cases} (P^{\tilde{\Phi}^{\mathrm{ct}}} \cup \{\overline{x} \mid \tilde{\Phi}^{\mathrm{tf}}(\psi^{\mathrm{ct}})\}, P^{\tilde{\Phi}^{\mathrm{cf}}}) & \text{if } Q = P \\ Q^{\tilde{\Phi}} & \text{otherwise.} \end{cases}$$

If φ is the INF sentence $\forall \overline{x}\ (\psi \Rightarrow \neg P(\overline{x}))$, then $\mathscr{I}_s^{\varphi}$ is defined by

$$Q^{\mathscr{I}_s^{\varphi}(\tilde{\Phi})} = \begin{cases} (P^{\tilde{\Phi}^{\mathrm{ct}}}, P^{\tilde{\Phi}^{\mathrm{cf}}} \cup \{\overline{x} \mid \tilde{\Phi}^{\mathrm{tf}}(\psi^{\mathrm{ct}})\}) & \text{if } Q = P \\ Q^{\tilde{\Phi}} & \text{otherwise.} \end{cases}$$

Of course, the desired property is that the symbolic INF propagator $\mathscr{I}_s^{\varphi}$ describes the non-symbolic INF propagator $\mathscr{I}^{\varphi}$. This is indeed the case.

Proposition 4.16. *$\mathscr{I}_s^{\varphi}$ describes $\mathscr{I}^{\varphi}$ for every INF sentence φ.*

Proof. We prove the case where φ is of the form $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$ and $L[\overline{x}]$ is a positive literal. The proof is similar in case $L[\overline{x}]$ is a negative literal.
Let E be a σ-structure and $\tilde{\Phi}$ a four-valued symbolic Σ-structure over σ. Let φ be the INF sentence $\forall \overline{x}\ (\psi \Rightarrow P(\overline{x}))$. We have to show that $E(\mathscr{I}_s^{\varphi}(\tilde{\Phi})) = \mathscr{I}^{\varphi}(E(\tilde{\Phi}))$. Therefore, we must prove that $Q^{E(\mathscr{I}_s^{\varphi}(\tilde{\Phi}))}(\overline{d}) = Q^{\mathscr{I}^{\varphi}(E(\tilde{\Phi}))}(\overline{d})$ for every predicate Q.
First assume $Q \neq P$. Then the following is a correct chain of equations.

$$Q^{\mathscr{I}^{\varphi}(E(\tilde{\Phi}))} = Q^{E(\tilde{\Phi})} = (Q^{\tilde{\Phi}})^E = (Q^{\mathscr{I}_s^{\varphi}(\tilde{\Phi})})^E = Q^{E(\mathscr{I}_s^{\varphi}(\tilde{\Phi}))} \tag{4.20}$$

The first and third equation follow from the definitions of $\mathscr{I}^{\varphi}$, respectively $\mathscr{I}_s^{\varphi}$, and the assumption that $Q \neq P$. The second and the fourth equation apply the definition of $E(\tilde{\Phi})$.
Now we show that $P^{E(\mathscr{I}_s^{\varphi}(\tilde{\Phi}))} = P^{\mathscr{I}^{\varphi}(E(\tilde{\Phi}))}$. From the definition of $\mathscr{I}^{\varphi}$ it follows that $P^{\mathscr{I}^{\varphi}(\tilde{I})^{\mathrm{cf}}} = P^{\tilde{I}^{\mathrm{cf}}}$ and $P^{\mathscr{I}^{\varphi}(\tilde{I})^{\mathrm{ct}}} = P^{\tilde{I}^{\mathrm{ct}}} \cup \{\overline{d}_x \mid \tilde{I}[\overline{x}/\overline{d}_x](\psi) \geq_p \mathbf{t}\}$ for any structure $\tilde{I}$. By Proposition 2.7, the latter equation rewrites to $P^{\mathscr{I}^{\varphi}(\tilde{I})^{\mathrm{ct}}} = P^{\tilde{I}^{\mathrm{ct}}} \cup \{\overline{x} \mid \psi^{\mathrm{ct}}\}^{\tilde{I}^{\mathrm{tf}}}$. Therefore,

$$\begin{aligned}
P^{\mathscr{I}^{\varphi}(E(\tilde{\Phi}))} &= \left(P^{E(\tilde{\Phi})^{\mathrm{ct}}} \cup \{\overline{x} \mid \psi^{\mathrm{ct}}\}^{E(\tilde{\Phi})^{\mathrm{tf}}}, P^{E(\tilde{\Phi})^{\mathrm{cf}}}\right) \\
&= \left(P^{E(\tilde{\Phi}^{\mathrm{ct}})} \cup \{\overline{x} \mid \psi^{\mathrm{ct}}\}^{E(\tilde{\Phi}^{\mathrm{tf}})}, P^{E(\tilde{\Phi}^{\mathrm{cf}})}\right) \\
&= \left((P^{\tilde{\Phi}^{\mathrm{ct}}})^E \cup \{\overline{x} \mid \tilde{\Phi}^{\mathrm{tf}}(\psi^{\mathrm{ct}})\}^E, (P^{\tilde{\Phi}^{\mathrm{cf}}})^E\right) \\
&= (P^{\mathscr{I}_s^{\varphi}})^E = P^{E(\mathscr{I}_s^{\varphi})}.
\end{aligned}$$

■

Proposition 4.16 implies that one can execute the propagation algorithm (Algorithm 4.2) using symbolic INF propagators in step 3 instead of non-symbolic ones. We refer to this symbolic version of the algorithm as the *symbolic propagation algorithm*.

Example 4.14. Consider the following INF sentences:

$$\forall x \forall y\ (\neg \mathit{Edge}(x,y) \Rightarrow \neg \mathit{Path}(x,y)) \tag{4.21}$$
$$\forall x \forall y\ (\mathit{Start}(y) \Rightarrow \neg \mathit{Path}(x,y)) \tag{4.22}$$
$$\forall x \forall y \forall z\ (\neg \mathit{Path}(x,y) \wedge \neg \mathit{Path}(x,z) \Rightarrow \mathit{Aux}(x,y,z)). \tag{4.23}$$

Let σ be a vocabulary containing predicates $\mathit{Graph}/2$ and $\mathit{Begin}/1$ and let $\tilde{\Phi}_0$ be the symbolic structure over σ assigning $(\{(x,y) \mid \mathit{Graph}(x,y)\}, \{(x,y) \mid \neg \mathit{Graph}(x,y)\})$ to Edge, $(\{x \mid \mathit{Begin}(x)\}, \{x \mid \neg \mathit{Begin}(x)\})$ to Start, $(\{(x,y) \mid \bot\}, \{(x,y) \mid \bot\})$ to Path and $(\{(x,y,z) \mid \bot\}, \{(x,y,z) \mid \bot\})$ to Aux. Applying $\mathscr{I}_s^{(4.21)}$ on $\tilde{\Phi}_0$ yields the symbolic structure $\tilde{\Phi}_1$ that assigns $(\{(x,y) \mid \bot\}, \{(x,y) \mid \bot \vee \neg \mathit{Graph}(x,y)\})$ to Path. Applying $\mathscr{I}_s^{(4.22)}$ on $\tilde{\Phi}_1$ produces symbolic structure $\tilde{\Phi}_2$ assigning $(\{(x,y) \mid \bot\}, \{(x,y) \mid \bot \vee \neg \mathit{Graph}(x,y) \vee \mathit{Begin}(y)\})$ to Path. Finally, the result of applying $\mathscr{I}_s^{(4.23)}$ to $\tilde{\Phi}_2$ assigns

$$\begin{aligned}
(\{(x,y,z) \mid &\bot \vee ((\bot \vee \neg \mathit{Graph}(x,y) \vee \mathit{Begin}(y)) \\
&\wedge (\bot \vee \neg \mathit{Graph}(x,z) \vee \mathit{Begin}(z)))\}, \{(x,y,z) \mid \bot\})
\end{aligned} \tag{4.24}$$

to Aux.

Observe that computing $\mathscr{I}_s^{\varphi}(\tilde{\Phi})$ takes time $\mathcal{O}(|\varphi| \cdot |\tilde{\Phi}|)$, while computing $\mathscr{I}^{\varphi}(\tilde{I})$ takes time $\mathcal{O}(|\tilde{I}|^{|\varphi|})$ (see Theorem 2.1). This indicates a possible benefit of using symbolic INF propagators instead of non-symbolic ones. However, this gain in efficiency does not come for free. One problem is that testing whether

a symbolic structure is a fixpoint of a symbolic INF propagator is undecidable, because it boils down to testing the equivalence of two FO sentences. As a direct consequence, when constructing a refinement sequence using symbolic INF propagators, it is undecidable whether a fixpoint has been reached. In some cases, a fixpoint cannot be reached, as illustrated by the following example.

Example 4.15. Let T be the theory from Example 4.9 and let σ be the vocabulary

$$\langle \{Action, Time\}, \{Prec(Action, Action)\}, \emptyset \rangle.$$

For every $n \in \mathbb{N}$, let $\psi_n[a,t]$ be the formula

$$\begin{aligned} &\exists a_0 \cdots \exists a_n \ (Prec(a_0, a) \wedge Prec(a_1, a_0) \ldots \wedge Prec(a_n, a_{n-1})) \\ &\wedge \forall t_0 \cdots \forall t_n \ (t_1 < t_0 \wedge \ldots \wedge t_n < t_{n-1} \Rightarrow t \leq t_0). \end{aligned}$$

Intuitively, $\psi_n[a,t]$ is true if there exists a chain of n preconditions of a and t is among the first n timepoints. Clearly, ψ_{n_1} and ψ_{n_2} are not logically equivalent if $n_1 \neq n_2$.
Now let $\tilde{\Phi}_0$ be the symbolic structure over σ, defined by

$$\begin{aligned} Do^{\tilde{\Phi}_0} &= (\{(a,t) \mid \bot\}, \{(a,t) \mid \bot\}), \\ Prec^{\tilde{\Phi}_0} &= (\{(a_1,a_2) \mid Prec(a_1,a_2)\}, \{(a_1,a_2) \mid \neg Prec(a_1,a_2)\}), \\ <^{\tilde{\Phi}_0} &= (\{(t_1,t_2) \mid t_1 < t_2\}, \{(t_1,t_2) \mid t_1 \geq t_2\}), \end{aligned}$$

and denote by V the set of symbolic INF propagators associated to $\mathrm{INF}(T)$. Then one can show that there exists for every $n \in \mathbb{N}$ a symbolic V-refinement sequence from $\tilde{\Phi}_0$ such that its last element assigns a query equivalent to $\{(a,t) \mid \psi_0 \vee \ldots \vee \psi_n\}$ to predicate Do^{cf}. This indicates that there exists no terminal V-refinement sequence form $\tilde{\Phi}_0$. Also, it can be proven that there exists no FO formula equivalent to the formula $(\bigvee_{0 \leq i \leq \infty} \psi_i)$, showing that even if other propagators are added to V, a terminal refinement sequence cannot be obtained.

Another problem concerning symbolic refinement sequences is the size of symbolic structures. The size of $\mathscr{I}_s^{\varphi}(\tilde{\Phi})$ is $\mathcal{O}(|\varphi| \cdot |\tilde{\Phi}|)$. As such, the size of the last element of a refinement sequence constructed using symbolic INF propagators is exponential in the length of the sequence, while for non-symbolic refinement sequences, the size of the last element is bounded by $\|\top^{\leq_p}\|$. The exponential growth of the symbolic structures can sometimes, but not always, be avoided by replacing the queries assigned by a structure by equivalent, but smaller queries. For example, (4.24) could be replaced by the shorter, equivalent pair of queries

$$\begin{aligned} &(\{(x,y,z) \mid (\neg Graph(x,y) \vee Begin(y)) \wedge (\neg Graph(x,z) \vee Begin(z))\}, \\ &\{(x,y,z) \mid \bot\}) \end{aligned}$$

For V and $\tilde{\Phi}_0$ as in Example 4.15, a strong simplification algorithm may reduce the size of the last element in a symbolic V-refinement sequence from $\tilde{\Phi}_0$ of length n to $\mathcal{O}(n^2)$.

We expect symbolic propagation to be useful in applications where precision is less important than efficiency, and where the evaluation $E(\tilde{\Phi})$ of the last structure $\tilde{\Phi}$ of a refinement sequence in σ-structure E need not be computed completely. Approximate query answering (Section 4.5.3) and grounding (Chapter 5) are two examples of such applications.

4.3.3 Implementing symbolic propagation

There are several issues when implementing the symbolic propagation algorithm. First of all, the symbolic INF propagators need to be applied in a good order as to minimize the amount of redundant work. Secondly, it follows from the discussion at the end of the previous section that a simplification algorithm for queries must be applied in order to avoid the exponential blow-up of structures in a refinement sequence. Thirdly, a stop criterion is needed to guarantee termination. Finally, in many applications it will be necessary to evaluate part of the computed symbolic structure. As such, an efficient algorithm to evaluate queries is needed. In this section, we present a basic solution to these problems. The algorithms we present below are implemented in the system GIDL.

The symbolic propagation algorithm

Our implementation of the symbolic propagation algorithm is presented by Algorithm 4.3. It includes a simple heuristic for choosing propagators and a basic stop criterion.

Algorithm 4.3 takes as input a theory T, a symbolic structure $\tilde{\Phi}$ and a constant $C \in \mathbb{N}$. It returns the last element of a $\{\mathscr{I}_s^\varphi \mid \varphi \in \mathrm{INF}(T)\}$-refinement sequence of length less than C. The algorithm maintains a queue Q of INF sentences from $\mathrm{INF}(T)$. The symbolic INF propagators corresponding to sentences on Q will be applied on $\tilde{\Phi}$.

In order to not schedule unnecessary propagators, a sentence $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$ such that $\tilde{\Phi}(\psi^{\mathrm{ct}})$ is equivalent to $\bot$ should not be pushed on Q. Indeed, if φ is such a sentence then the queries assigned by $\mathscr{I}_s^\varphi(\tilde{\Phi})$ are equivalent to the ones assigned by $\tilde{\Phi}$. However, it is undecidable whether $\tilde{\Phi}(\psi^{\mathrm{ct}})$ is equivalent to $\bot$. Algorithm 4.3 overestimates the set of useful propagators: it may schedule a sentence $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$ as long as $\tilde{\Phi}(\psi^{\mathrm{ct}})$ contains at least one positively (negatively) occurring subformula that is not equal to $\bot$ ($\top$). Initially, all sentences that satisfy this criterion are scheduled (lines 1 to 5).

In the while loop, the algorithm implicitly constructs a refinement sequence: the successive values of $\tilde{\Phi}$ form such a sequence. The variable n stores the current length of the sequence. As long as there are scheduled INF sentences and the length of the refinement sequence does not exceed threshold C, the propagator corresponding to the first sentence on Q is applied. This is done in line 10: q stores the value of P in $(\mathscr{I}_s^{\forall \overline{x}\ (\psi \Rightarrow P(\overline{x}))}(\tilde{\Phi}))^{\mathrm{ct}}$. Next, a simplification algorithm is applied to simplify q. We discuss such an algorithm below.

If q is not equivalent to the current value of P in $\tilde{\Phi}$ (according to the necessarily incomplete equivalence check `equivalent`) and q is "acceptable", then $\tilde{\Phi}$ is up-

Algorithm 4.3: Symbolic propagation

Input: A theory T over Σ, a symbolic Σ-structure $\tilde{\Phi}$ and a constant $C \in \mathbb{N}$.

$Q := \emptyset$;
for $P \in \Sigma^{\mathrm{tf}}_{\mathrm{pred}}$ **do**
  if $P^{\tilde{\Phi}^{\mathrm{tf}}}$ *is not of the form* $\{\overline{x} \mid \bot\}$ **then**
    for *all* $(\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])) \in \mathrm{INF}(T)$ *such that* P *occurs in* ψ^{ct} **do**
      Q.push$(\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}]))$;
$n := 0$;
while $Q \neq \emptyset$ *and* $n < C$ **do**
  $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}]) := Q$.pop();
  if $L[\overline{x}]$ *is a positive literal* $P(\overline{x})$ **then**
    $q := P^{\tilde{\Phi}^{\mathrm{ct}}} \cup \{\overline{x} \mid \tilde{\Phi}^{\mathrm{tf}}(\psi^{\mathrm{ct}})\}$;
    $q :=$ `simplify(`q`)`;
    if *(not* `equivalent(`q`,`$P^{\tilde{\Phi}^{\mathrm{ct}}}$`)` *) and* `acceptable(`$q$`,`$P^{\tilde{\Phi}^{\mathrm{ct}}}$`)` **then**
      $n := n + 1$;
      $P^{\tilde{\Phi}^{\mathrm{ct}}} := q$;
      for *all* $(\forall \overline{y}\ (\chi \Rightarrow L'[\overline{y}])) \in (\mathrm{INF}(T) \setminus Q)$ *such that* P *occurs positively in* χ^{ct} **do**
        Q.push$(\forall \overline{y}\ (\chi \Rightarrow L'[\overline{y}]))$;
  else
    `...  // Similar code when` $L[\overline{x}]$ `is a negative literal.`
return $\tilde{\Phi}$;

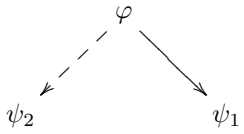


Figure 4.2: Graph representation of the formula $\varphi \rightarrowtriangle \psi_1; \psi_2$. The solid arrow points to the 'if' case, the dashed arrow to the 'else' case.

dated, i.e., the refinement sequence is extended with $\mathscr{I}_s^{\forall \overline{x}\ (\psi \Rightarrow P(\overline{x}))}(\tilde{\Phi})$. Finally, all INF sentences $\forall \overline{y}\ (\chi \Rightarrow L'[\overline{y}])$ of $\mathrm{INF}(T)$ that are not yet on Q and such that the value of χ^{ct} in the updated structure is different from its value in the previous structure, are pushed on Q (line 16).
The condition $n < C$ of the while loop and the conditions in line 12 form the stop criterion of the algorithm. The former condition ensures termination of the algorithm since in each iteration of the while loop, a sentence is popped from the Q and n is increased if a new sentence is pushed on Q. Since the algorithm schedules all propagators that can possibly change $\tilde{\Phi}$, the first condition in line 12 enables detecting that the constructed refinement sequence is terminal. Indeed, Q will be emptied if `equivalent` detects that all scheduled propagators $\mathscr{I}_s^{\varphi}$, $\mathscr{I}_s^{\tilde{\Phi}}$ and $\tilde{\Phi}$ are equivalent. The second condition in line 12 may rule out the application of propagators that produce a query that is unacceptable to some criterion. For example, it could ensure the queries do not become larger than some fixed value, as to certainly avoid the exponential growth of $\tilde{\Phi}$. We discuss different possibilities for the function `acceptable` in Section 5.4.

First-order binary decision trees and diagrams

In a practical implementation of Algorithm 4.3, the simplification algorithm (line 11) is of vital importance. Goubault (1995) investigated such a simplification algorithm. His algorithm uses *first-order binary decision diagrams* to represent and simplify formulas. We show in this section that such a representation can be applied without too much overhead when applying symbolic INF propagators. Moreover, using binary decision diagrams leads to extra benefits: we obtain a cheap equivalence check for queries and an elegant algorithm to evaluate queries. The latter is needed to compute the value of a symbolic structure over σ in a σ-structure E.
We borrow the definition of first-order BDDs from Goubault (1995). Let φ, ψ_1 and ψ_2 be three formulas. The ternary *if-then-else operator* is denoted by '$\rightarrowtriangle$', and defined by $\varphi \rightarrowtriangle \psi_1; \psi_2 \ := \ (\varphi \wedge \psi_1) \vee (\neg\varphi \wedge \psi_2)$. The formula $\varphi \rightarrowtriangle \psi_1; \psi_2$ is also represented by the graph shown in Figure 4.2.

Definition 4.10 (Goubault (1995))**.** *FO binary decision trees* (BDTs) and *kernels* are defined by simultaneous induction:

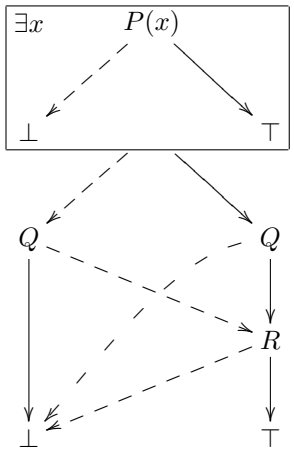


Figure 4.3: A first-order BDD representing the formula $((\exists x\ P(x)) \wedge Q \wedge R) \vee ((\forall x\ \neg P(x)) \wedge \neg Q \wedge R)$.

- An atom is a kernel;
- If φ is a BDT and x a variable, then $(\exists x\ \varphi)$ is a kernel;
- $\top$ and $\bot$ are BDTs;
- If φ is a kernel and ψ_1 and ψ_2 are BDTs, then $\varphi \rightarrowtriangle \psi_1; \psi_2$ is a BDT.

If the graphical representation for if-then-else formulas is used, a BDT corresponds to a tree with kernels as nodes. Each non-atomic kernel contains a BDT itself.
Goubault (1995) showed that for every FO formula φ there exists a BDT φ' such that φ and φ' are equivalent. In an actual implementation, *sharing*, *reducing* and *ordering* are applied to obtain a simplified and compact representation of BDTs. Such representations are called *reduced ordered binary decision diagrams* (BDDs). Sharing means that isomorphic subtrees are stored at the same address in memory. Reducing involves exhaustively replacing subtrees of the form $\varphi \rightarrowtriangle \psi; \psi$ by ψ. A BDT φ is ordered if the kernels appear in some fixed order on every path in the graph representation of φ. Figure 4.3 shows a BDD.
As mentioned above, there are several important benefits of using BDDs to represent formulas in queries:

- An implementation of Algorithm 4.3 using BDDs allows us to use the simplification algorithm for BDDs of Goubault (1995) in line 11.
- As explained above, to detect that a symbolic refinement sequence is terminal, one needs to check equivalence of queries, i.e., equivalence of the

formulas in queries (see line 12 of Algorithm 4.3). Often, the BDDs representing two equivalent formulas will be equal.[18] Hence, a cheap (but necessarily incomplete) equivalence check for two bounds consists of checking the syntactic equality of the two BDDs representing them. Since equal BDDs are stored at the same address, this check is done in constant time.

- As we will show below, computing the value of a query $\{\overline{x} \mid \varphi\}$ in a structure can easily be implemented directly on a BDD representation of φ. Computing the value of queries is necessary to compute the value of (part of) a symbolic structure. As we will see in Section 5.4, this is one of the main operations performed by a grounder.

On the other hand, using BDDs does not result in too much overhead when computing a symbolic refinement sequence. If φ, ψ and $\chi[x, \overline{y}]$ are represented by BDDs, then a BDD representing $(\neg\varphi)$, $(\exists x\ \varphi)$, $(\forall x\ \varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $(\chi[x/x', \overline{y}])$ can be computed efficiently (Bryant, 1986; Goubault, 1995). This implies that the result of applying a symbolic INF propagator on a symbolic structure $\tilde{\Phi}$ can be computed efficiently, even if $\tilde{\Phi}$ is represented using BDDs. If BDDs are used to represent the queries assigned by $\tilde{\Phi}$, line 10 of Algorithm 4.3 can be implemented in linear time in the size of $\tilde{\Phi}$. If we use Goubault's simplification algorithm for BDDs for implementing line 11, the worst case complexity of this step is non-elementary in the size of q (Goubault, 1995). That is, for any $n \in \mathbb{N}$, there exists a q such that computing `simplify(`q`)` takes more time than $m^{|q|}$, where

$$m = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n \text{ times}}.$$

It may seem that the complexity of the simplification method limits the practical applicability of Algorithm 4.3. However, if `acceptable` is designed so that large BDDs (usually) do not pass the test in line 12, the simplification method is rarely applied on large BDDs. In our experiments with the system GIDL, the running time of the propagation algorithm was negligible compared to the running time of the whole system (see Section 5.4).

Simplification by partial evaluation

In a context where the σ-structure E that will be used to evaluate symbolic structures is fixed and known, the queries assigned by a symbolic structure $\tilde{\Phi}$ can be simplified by partially evaluating them in E. Specifically, assume $\{\overline{x} \mid \varphi\}$ is a query assigned by $\tilde{\Phi}$ and $\psi[\overline{y}]$ is a non-atomic subformula of φ with n free variables. Let Q/n be a new predicate and let $\sigma' = \sigma \cup \{Q\}$. Let E' be the expansion of E to σ' defined by $Q^{E'} = \{\overline{y} \mid \psi\}^E$ and denote by φ' the result of replacing ψ by $Q(\overline{y})$ in φ. Note that φ' is smaller than φ. If $\tilde{\Phi}'$ is the symbolic structure obtained from $\tilde{\Phi}$ by replacing the query $\{\overline{x} \mid \varphi\}$ by $\{\overline{x} \mid \varphi'\}$, then the

[18] For propositional BDDs, this is always the case.

largest query assigned by $\tilde{\Phi}'$ is not larger than the largest query assigned by $\tilde{\Phi}$, while the non-symbolic structure $E(\tilde{\Phi})$ is equal to $E'(\tilde{\Phi}')$.
A special case arises when the substituted subformula ψ has no free variables. Then introducing a new predicate Q is not needed. Indeed, in this case $\{() \mid \psi\}^E$ is the empty set if $E \not\models \psi$ and the set containing the empty tuple of $E \models \psi$. In the former case ψ can be replaced by $\bot$, in the latter case by $\top$.

Example 4.16. Assume one of the queries assigned by $\tilde{\Phi}$ is the query $\{x \mid \exists x\ P(x)\}$. Then this query can be simplified to $\{x \mid \bot\}$ if $P^E = \emptyset$ and by $\{x \mid \top\}$ otherwise.

Querying

To compute (part of) the value of a symbolic Σ-structure $\tilde{\Phi}$ over σ in a σ-structure E, answers in E to queries $\{\overline{x} \mid \varphi\}$ must be computed, i.e., tuples $\overline{d}$ of domain elements must be searched such that $E[\overline{x}/\overline{d}] \models \varphi$. We now show that answering a query $\{\overline{x} \mid \varphi\}$ can be done directly on a BDD representation of φ by a simple backtracking algorithm.
The idea is to traverse the BDD, starting from the root, and trying to end up in the leaf $\top$. At each inner node $\psi[\overline{y}] \rightarrowtriangle \psi_1; \psi_2$, the free variables in that node are replaced by domain constants $\overline{d}_y$. If $E \models \psi[\overline{y}/\overline{d}_y]$, the algorithm continues via ψ_1, otherwise via ψ_2. If it ends up in $\bot$, it backtracks. If on the other hand, it ends up in $\top$, the performed substitutions constitute an answer for φ.
The function `query` implements the sketched query algorithm. It gets a query $\{\overline{x} \mid \varphi\}$ as input and returns a variable assignment $[\overline{x}/\overline{d}]$ such that $E[\overline{x}/\overline{d}] \models \varphi$. If no such substitution exists, it returns FAIL. This algorithm can easily be adapted to return all answers to $\{\overline{x} \mid \varphi\}$ instead of just one.
In lines 11 and 15, the algorithm needs to find tuples $\overline{d}$ such that respectively $E[\overline{v}/\overline{d}] \models \psi$ and $E[\overline{v}/\overline{d}] \not\models \psi$. If $\psi[\overline{v}]$ is an atom $P(\overline{v})$, this can be implemented by consulting the table P^E. If ψ is a kernel $(\exists x\ \chi[x, \overline{v}])$, function `query` can be applied recursively to find the tuples. Indeed, any answer $(d', \overline{d})$ to $\{(x, \overline{v}) \mid \chi\}$ provides a tuple $\overline{d}$ such that $E[\overline{v}/\overline{d}] \models \psi$. Vice versa, $E[\overline{v}/\overline{d}] \not\models \psi$ if $\{(x, \overline{v}) \mid \chi\}$ has no answers.
We illustrate the query algorithm on an example.

Example 4.17. Let $\varphi[x, y]$ be the BDD shown in figure 4.4, and let E be a structure with domain $\{a, b\}$ and $P^E = \{b\}$, $R^E = \{\}$ and $Q^E = \{(b, b)\}$. To find an answer for $\{(x, y) \mid \varphi\}$ in E, the query algorithm starts at the root $P(x)$. Since none of its children are equal to $\bot$, both $[x/a]$ and $[x/b]$ are possibly tried. Assume $[x/a]$ is tried first. Because $a \notin P^E$, the algorithm continues with node $R(a) \rightarrowtriangle \top; \bot$. Because the "else" child of this node is $\bot$ and $a \notin R^E$, the algorithm returns to the root and tries assignment $[x/b]$. Since $b \in P^E$, it goes to node $Q(b, y) \rightarrowtriangle \top; \bot$. Since the "else" child of this node is $\bot$, the algorithm tries those assignments $[x/d]$ such that $(b, y/d) \in Q^E$. Thus, only $[y/b]$ is tried and finally, answer $[x/b, y/b]$ is returned.

Function `query`$(E,\{\overline{x} \mid \varphi[\overline{y}]\})$

```
1  z̄ := x̄ \ ȳ;
2  if z̄ ≠ ∅ then
3      θ := query(E,{ȳ | φ[ȳ]});
4      if θ = FAIL then return FAIL;
5      else
6          Choose a tuple d̄_z ∈ 𝔰(z̄)^E;
7          return θ[z̄/d̄_z];
8  else if φ = ⊤ then
9      return the empty variable assignment;
10 else if φ = ψ[v̄] ⇁ ψ_1; ⊥ then
11     for every tuple d̄ such that E[v̄/d̄] ⊨ ψ do
12         θ := query(E,{x̄ \ v̄ | ψ_1[v̄/d̄]});
13         if θ ≠ FAIL then return θ[v̄/d̄]
14 else if φ = ψ[v̄] ⇁ ⊥; ψ_2 then
15     for every tuple d̄ such that E[v̄/d̄] ⊭ ψ do
16         θ := query(E,{x̄ \ v̄ | ψ_2[v̄/d̄]});
17         if θ ≠ FAIL then return θ[v̄/d̄];
18 else if φ is of the form ψ[v̄] ⇁ ψ_1; ψ_2 then
19     for every tuple d̄ ∈ 𝔰(v̄)^E do
20         if E[v̄/d̄] ⊨ ψ then θ := query(E,{x̄ \ v̄ | ψ_1[v̄/d̄]});
21         else θ := query(E,{x̄ \ v̄ | ψ_2[v̄/d̄]});
22         if θ ≠ FAIL then return θ[v̄/d̄];
23 return FAIL;
```

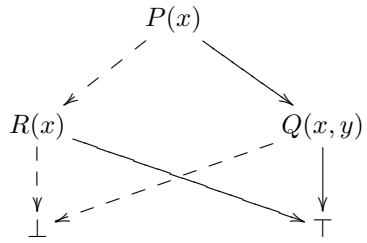


Figure 4.4: A BDD representing the formula $(P(x) \wedge Q(x,y)) \vee (\neg P(x) \wedge R(x))$

4.4 Propagation for extended first-order logic

In the previous section we developed a concrete algorithm for propagation in function-free FO theories. We now investigate how to extend this algorithm to FO($\cdot$). To this end, a normal form like INF is developed for FO($\cdot$), as well as propagators associated to this normal form. Furthermore, we explain how to reduce each FO($\cdot$) theory to the proposed normal form. Finally, we investigate whether the benefits of INF propagators, namely the representation by a definition and the possibility of symbolic execution, carry over to the INF propagators for FO($\cdot$).

4.4.1 Functions

There exists a straightforward extension of the propagation method to FO with functions. Indeed, in Section 2.1.4 we showed how to transform an FO theory T with functions into an "equivalent" function-free theory: first T is converted to TNF, next every atom of the form $F(\overline{x}) = y$ is replaced by $\mathscr{G}_F(\overline{x}, y)$ and finally, the sentences

$$\forall \overline{x} \exists y \; \mathscr{G}_F(\overline{x}, y) \tag{4.25}$$

$$\forall \overline{x} \forall y_1 \forall y_2 \; (\mathscr{G}_F(\overline{x}, y_1) \wedge \mathscr{G}_F(\overline{x}, y_2) \Rightarrow y_1 = y_2) \tag{4.26}$$

are added for every function symbol F. Denote the resulting theory by T'. For every model M of T, the structure $\mathscr{G}(M)$ is a model of T'. Vice versa, every model M' of T' is function consistent and $\mathscr{G}^{-1}(M')$ is a model of the original theory T.
The following propagation method emerges. For an input theory T over Σ and structure $\tilde{I}$, first convert T to an equivalent function-free theory T'. Then apply the propagation algorithm (Algorithm 4.2) to T' and $\mathscr{G}(\tilde{I})$ to obtain a more precise $\mathscr{G}(\Sigma)$-structure $\tilde{J}$. Finally, if $\tilde{J}$ is function consistent, return $\mathscr{G}^{-1}(\tilde{J})$. Since this method only relies on INF propagators, the propagation can be represented by a definition and it can be executed symbolically.
The question arises whether the computed structure $\tilde{J}$ is function consistent, i.e., whether $\tilde{J}$ can be converted to a four-valued Σ-structure. In general, this is not the case. However, $\tilde{J}$ is certainly function consistent if the refinement sequence in step 3 of Algorithm 4.2 is terminal, i.e., if $\tilde{J} = \lim_{\mathscr{I}(\mathrm{INF}(T'))}(\mathscr{G}(\tilde{I}))$. More precisely, we have the following proposition.

Proposition 4.17. *Let T be the TNF theory consisting of the sentences* (4.25) *and* (4.26) *and let $\tilde{I}$ be a $\mathscr{G}(\Sigma)$-structure. Then $\lim_{\mathscr{I}(\mathrm{INF}(T))}(\tilde{I})$ is function consistent.*

Proof. See Appendix A. ■

A note on precision

Recall that there are two methods to move functions outside predicate: either (2.13) is applied and an existential quantifier is introduced, or (2.14) is

applied and a universal quantifier is introduced. The sentences produced by both methods are logically equivalent, but may lead to different, and incomplete, propagation.

Example 4.18. Let φ be the sentence $P \Leftrightarrow Q(C)$, where C is a constant. Let D be the domain $\{a, b\}$ and let $\tilde{I}$ and $\tilde{J}$ be three-valued structures with domain D that assign

$$C^{\tilde{I}} = C^{\tilde{J}} = \{a, b\}$$
$$P^{\tilde{I}} = P^{\tilde{J}} = (\emptyset, \emptyset)$$
$$Q^{\tilde{I}} = (\{a, b\}, \emptyset)$$
$$Q^{\tilde{J}} = (\emptyset, \{a, b\})$$

In both $\tilde{I}$ and $\tilde{J}$, P is unknown and every domain element is still a possible value for C. In $\tilde{I}$, Q is true on the whole domain, in $\tilde{J}$ it is false on the whole domain. Because in every 2-valued structure, constant C is mapped to a domain element, it is clear that $Q(C)$ is true, respectively false, in every two-valued structure I approximated by $\tilde{I}$, respectively $\tilde{J}$. Therefore P is true in $\mathscr{O}^{\varphi}(\tilde{I})$ and false in $\mathscr{O}^{\varphi}(\tilde{J})$.
The two mentioned conversions of φ to function-free FO both contain the sentences

$$\exists x \; \mathscr{G}_C(x) \tag{4.27}$$
$$\forall x_1 \forall x_2 \; (\mathscr{G}_C(x_1) \wedge \mathscr{G}_C(x_2) \Rightarrow x_1 = x_2), \tag{4.28}$$

and they either contain

$$P \Leftrightarrow (\forall x \; (\mathscr{G}_C(x) \Rightarrow Q(x))) \tag{4.29}$$

or

$$P \Leftrightarrow (\exists x \; (\mathscr{G}_C(x) \wedge Q(x))). \tag{4.30}$$

We now consider the propagation of these sentences on $\tilde{I}$ and $\tilde{J}$. Note that in both structures $\mathscr{G}_C$ is assigned the value $(\emptyset, \emptyset)$, i.e., it is unknown on all domain elements.
One can check that applying $\mathscr{O}^{(4.27)}$ and $\mathscr{O}^{(4.28)}$ yield no propagation on $\tilde{I}$ and $\tilde{J}$. Since $\forall x \; (\mathscr{G}_C(x) \Rightarrow Q(x))$ evaluates to true in $\tilde{I}$, it is also true in all $M \geq_p \tilde{I}$. Therefore, $P^{\mathscr{O}^{(4.29)}(\tilde{I})} = \mathbf{t}$. On the other hand, there exists an $M \geq_p \tilde{I}$ such that $M \not\models \exists x \; (\mathscr{G}_C(x) \wedge Q(x))$. It follows that $P^{\mathscr{O}^{(4.30)}(\tilde{I})} = \mathbf{u}$. This shows that $\mathscr{L}_{\{(4.27),(4.28),(4.29)\}}$ and $\mathscr{L}_{\{(4.27),(4.28),(4.30)\}}$ are not the same propagator, and that $\mathscr{L}_{\{(4.27),(4.28),(4.30)\}}(\tilde{I}) <_p \mathscr{O}^{\varphi}(\tilde{I})$. Note that this implies that applying the propagation algorithm on input theory $\{(4.27), (4.28), (4.30)\}$ and structure $\tilde{I}$ leads to incomplete propagation. Similarly, one can show that $\mathscr{O}^{(4.29)}(\tilde{J})(Q) = \mathbf{u} \neq \mathbf{f} = \mathscr{O}^{(4.30)}(\tilde{J})(Q)$. As such, also $\mathscr{L}_{\{(4.27),(4.28),(4.29)\}}$ is less precise than $\mathscr{O}^{\varphi}$.

Example 4.18 shows that applying the transformations (2.13) or (2.14) separately to eliminate function symbols lead to incomplete propagation. Yet, one can show that for φ as in Example 4.18, applying both transformations simultaneously yields complete propagation, i.e., $\mathscr{L}_{\{(4.27),(4.28),(4.29),(4.30)\}} = \mathscr{O}^{\varphi}$. The next example shows that even applying both rewritings may lead to incomplete propagation.

Example 4.19. Let φ be the sentence $P \Leftrightarrow \forall x\ Q(C,x)$ and $\tilde{I}$ a three-valued structure with domain $\{a,b\}$ defined by $P^{\tilde{I}} = (\emptyset,\emptyset)$, $C^{\tilde{I}} = \{a,b\}$ and $Q^{\tilde{I}} = (\{(a,a),(b,b)\},\{(a,b),(b,a)\})$. One can verify that $P^{\mathscr{O}^{\varphi}(\tilde{I})} = \mathbf{f}$. On the other hand, applying both rewritings to eliminate C yields the sentences (4.27), (4.28),

$$P \Leftrightarrow \forall x \forall y\ (\mathscr{G}_C(y) \Rightarrow Q(y,x)) \tag{4.31}$$

and

$$P \Leftrightarrow \forall x \exists y\ (\mathscr{G}_C(y) \wedge Q(y,x)). \tag{4.32}$$

Both $\forall x \forall y\ (\mathscr{G}_C(y) \Rightarrow Q(y,x))$ and $\forall x \exists y\ (\mathscr{G}_C(y) \wedge Q(y,x))$ evaluate to unknown in $\tilde{I}$, hence P is unknown in $\mathscr{L}_{\{(4.27),(4.28),(4.31),(4.32)\}}(\tilde{I})$. We conclude that $\mathscr{L}_{\{(4.27),(4.28),(4.31),(4.32)\}}$ is less precise than $\mathscr{O}^{\varphi}$.

The next example shows that it is unlikely to find a complete propagation method with polynomial data-complexity for ENF theories if functions are allowed in ENF sentences.

Example 4.20. Let T be the ENF theory consisting of the sentences

$$\forall x \forall y\ (x = y \Leftrightarrow F(x) = F(y)),$$
$$\forall x \forall y\ (E_1(x,y) \Leftrightarrow E_2(F(x),F(y))).$$

Let $\tilde{I}$ a three-valued structure with domain D that is two-valued on E_1 and E_2 and assigns D to $F^{\tilde{I}}(d)$ for every $d \in D$. That is, F is completely unknown. Then $\mathscr{O}^T(\tilde{I})$ is consistent iff the graphs represented by E_1 and E_2 are isomorphic. At the moment, it is an open question whether a polynomial algorithm exists to solve the latter problem (Garey and Johnson, 1990).

Partial functions

Polynomial propagation for partial functions can be obtained in a similar way to propagation for functions. There are only two differences. First, the method used to convert a theory containing partial functions to TNF is dictated by the semantics we introduced in Section 3.1: only cautious rewrites can be applied. Secondly, while converting a theory to TNF, for a partial function F only the sentence (4.26) is added, instead of both (4.25) and (4.26).

4.4.2 Sorts and arithmetic

Only two small difficulties arise when subsorts and arithmetic are allowed in the input for the propagation algorithm. The first one concerns the union of

queries. For example, assume that, as in Example 4.1, *Selected* is a predicate of sort MC, which denotes the union of the sorts *Course* and *Module*, and consider the following INF sentence

$$\forall c\ (\top \Rightarrow \mathit{Selected}(c)), \tag{4.33}$$

where c is a variable of sort *Course*. This sentence expresses that all courses are selected. Let $\tilde{\Phi}$ a four-valued symbolic structure that assigns $(\{x \mid \bot\}, \{x \mid \bot\})$ to *Selected*. Here, x is a variable of sort MC. Applying the symbolic INF propagator $\mathscr{I}_s^{(4.33)}$ on $\tilde{\Phi}$ produces $(\{x \mid \bot\} \cup \{c \mid \top\}, \{x \mid \bot\})$ as queries assigned to *Selected*. In this case it is wrong to apply our definition of '$\cup$' and substitute $\{x \mid \bot\} \cup \{c \mid \top\}$ by $\{y \mid \bot \vee \top\}$, where y is a new variable of sort MC. Instead, $\{x \mid \bot\} \cup \{c \mid \top\}$ should be replaced by $\{y \mid \bot \vee (\mathit{Course}_{\mathrm{pred}}(y) \wedge \top)\}$. In general, we define

$$\begin{aligned}\{(x_1, \ldots, x_n) \mid \varphi\} \cup \{(y_1, \ldots, y_n) \mid \psi\} = \{(z_1, \ldots, z_n) \mid \\ (\mathfrak{s}(x_1)_{\mathrm{pred}}(z_1) \wedge \ldots \wedge \mathfrak{s}(x_n)_{\mathrm{pred}}(z_n) \wedge \varphi[x_1/z_1, \ldots, x_n/z_n]) \\ \vee (\mathfrak{s}(y_1)_{\mathrm{pred}}(z_1) \wedge \ldots \wedge \mathfrak{s}(y_n)_{\mathrm{pred}}(z_n) \wedge \psi[y_1/z_1, \ldots, y_n/z_n])\}\end{aligned}$$

The second difficulty is that structures over a vocabulary Σ containing $\mathfrak{Int}$ contain an infinite domain. Even propagation from a finite[19] Σ-structure may produce an infinite one. For example, if $P(\mathfrak{Int}) \in \Sigma_{\mathrm{pred}}$, T is a theory containing the INF sentence $\varphi := \forall x\ (\top \Rightarrow P(x))$, where $\mathfrak{s}(x) = \mathfrak{Int}$, and $\tilde{I}$ is a finite structure assigning $P^{\tilde{I}} = (\emptyset, \emptyset)$, then $P^{\mathscr{I}^\varphi(\tilde{I})} = (\mathbb{Z}, \emptyset)$. Hence $\mathscr{I}^\varphi(\tilde{I})$ is infinite in this case. We conclude that the non-symbolic propagation method is not always applicable in practice when a vocabulary Σ containing $\mathfrak{Int}$ is considered. Observe that there is no problem if $\mathfrak{Int} \notin \mathfrak{s}(P)$ and $\mathfrak{Int} \notin \mathfrak{s}(F)$ for every predicate symbol P and function symbol F in Σ that is not "built-in", i.e., is not among '$<$', '$+$', '$\cdot$', 'abs', etc. For such a vocabulary, propagation from a finite structure always produces a finite structure or an inconsistent one. Also, including $\mathfrak{Int}$ in a vocabulary does not lead to problems for symbolic propagation, since each symbolic structure has finite size. Of course, evaluating a symbolic structure $\tilde{\Phi}$ in a structure E might produce an infinite structure.

4.4.3 Aggregates

To extend the propagation method to FO(AGG), the definition of INF sentences is extended to include aggregates. As for FO, a propagator with polynomial time data-complexity is associated to each of these sentences. Next, it is shown that every FO(AGG) theory over a vocabulary Σ can be converted to a Σ-equivalent theory of INF sentences. To represent propagation on FO(AGG) theories as a definition, the definition of φ^{ct} is extended to formulas φ that may contain aggregates. It then also follows that symbolic propagation for INF sentences containing aggregates is possible.

[19] Recall that a structure containing the integers *finite* if the interpretation of all symbols that are not "built-in" is finite (see Section 3.3).

In this section, we assume that $\mathfrak{Int} \notin \mathfrak{s}(P)$ and $\mathfrak{Int} \notin \mathfrak{s}(F)$ for every predicate symbol P and function symbol F that is not "built-in". This guarantees that propagating a finite structure yields a finite structure or an inconsistent one.

Definition 4.11. A FO(AGG) sentence φ is in *implicational normal form* (INF) if it is of the form $\forall \overline{x}(\psi \Rightarrow L[\overline{x}])$, where $L[\overline{x}]$ is a literal that does not contain an aggregate and ψ is a formula in TNF. The result of applying the INF propagator $\mathscr{I}^{\varphi}$ associated to φ on a three-valued structure $\tilde{I}$ is defined as in definition 4.3. If $\tilde{I}$ is strictly four-valued, then we define $\mathscr{I}^{\varphi}(\tilde{I}) = \top^{\leq_p}$.

Proposition 4.18. *For every INF sentence φ, $\mathscr{I}^{\varphi}$ is a monotone propagator with polynomial time data-complexity.*

Proof. The proof is similar to the proof of Proposition 4.8. The fact that the data-complexity is in **P** follows from Proposition 3.7. ■

We now show that every FO(AGG) theory T over vocabulary Σ can be converted to a Σ-equivalent theory containing only INF sentences. Similarly as for FO theories, we present a conversion in several steps. However, complete propagation is preserved in none of the steps. The following example indicates that even for very simple theories, complete polynomial time propagation is impossible if $\mathbf{P} \neq \mathbf{NP}$.

Example 4.21. Let T be the theory containing the sentence $\textsc{sum}\{x \mid P(x)\} = n$, where n is a natural number. Let $\tilde{I}$ be a finite structure with domain $D \subset \mathbb{N}$ such that $P^{\tilde{I}}(d) = \mathbf{u}$ for every $d \in D$. Then $\mathscr{O}^T(\tilde{I}) \neq \top^{\leq_p}$ iff $\sum_{d \in V} d = n$ for some subset $V \subseteq D$. Deciding whether such a subset exists is **NP**-complete (see, e.g., Sipser, 2005). Hence if $\mathbf{P} \neq \mathbf{NP}$, $\mathscr{O}^T$ cannot be implemented by a polynomial time algorithm.

Definition 4.12. A FO(AGG) sentence φ is in *equivalence normal form* (ENF) if φ is an FO sentence in ENF or φ is of one of the following forms, where $Q \neq P$:

- $\forall \overline{x} \forall z\ (P(\overline{x}, z) \Leftrightarrow \textsc{f}\{\overline{y} \mid Q(\overline{x}, \overline{y})\} \leq z)$,
- $\forall \overline{x} \forall z\ (P(\overline{x}, z) \Leftrightarrow \textsc{f}\{\overline{y} \mid \neg Q(\overline{x}, \overline{y})\} \leq z)$,
- $\forall \overline{x} \forall z\ (P(\overline{x}, z) \Leftrightarrow \textsc{f}\{\overline{y} \mid Q(\overline{x}, \overline{y})\} \geq z)$,
- $\forall \overline{x} \forall z\ (P(\overline{x}, z) \Leftrightarrow \textsc{f}\{\overline{y} \mid \neg Q(\overline{x}, \overline{y})\} \geq z)$.

Proposition 4.19. *Every* FO(AGG) *theory T over Σ can be rewritten to a Σ-equivalent theory in ENF.*

Proof. First rewrite T to TNF as indicated below Definition 3.14. Next, replace every subformula of the form $\textsc{f}(V) = x$ by the conjunction $(\textsc{f}(V) \leq x \wedge \textsc{f}(V) \geq x)$. Finally, Algorithm 4.1 can be applied to rewrite to an ENF theory. ■

φ	$\text{cs}(\varphi)$	$\text{cl}(\varphi)$
$\text{CARD}(\{\overline{x} \mid \psi\} \setminus \overline{y}) \leq z$	$\bot$	$\top$
$\text{SUM}(\{(n, \overline{x}) \mid \psi\} \setminus (n', \overline{y})) \leq z$	$n' < 0$	$n' > 0$
$\text{PROD}(\{(n, \overline{x}) \mid \psi\} \setminus (n', \overline{y})) \leq z$	$z < 0 \wedge n' > 1$	$z < 0 \wedge n' < 0$
$\text{MIN}(\{(n, \overline{x}) \mid \psi\} \setminus (n', \overline{y})) \leq z$	$\bot$	$\bot$
$\text{MAX}(\{(n, \overline{x}) \mid \psi\} \setminus (n', \overline{y})) \leq z$	$\bot$	$n' > z$
$\text{CARD}(\{\overline{x} \mid \psi\} \setminus \overline{y}) \geq z$	$\bot$	$\top$
$\text{SUM}(\{(n, \overline{x}) \mid \psi\} \setminus (n', \overline{y})) \geq z$	$n' < 0$	$n' > 0$
$\text{PROD}(\{(n, \overline{x}) \mid \psi\} \setminus (n', \overline{y})) \geq z$	$z > 0 \wedge n' < 0$	$z > 0 \wedge n' > 1$
$\text{MIN}(\{(n, \overline{x}) \mid \psi\} \setminus (n', \overline{y})) \geq z$	$n' < z$	$\bot$
$\text{MAX}(\{(n, \overline{x}) \mid \psi\} \setminus (n', \overline{y})) \geq z$	$\bot$	$\bot$

Table 4.2: Definition of the formulas $\text{cs}(\varphi)$ and $\text{cl}(\varphi)$.

From FO(AGG) to INF

Similarly as for FO sentences in ENF, a set $\text{INF}(\varphi)$ of INF sentences is associated to each ENF sentence φ containing an aggregate. Our definition of this set is inspired by the propagation algorithms for propositional aggregates in the model generator MiniSAT(ID) (Mariën, 2009). Intuitively, the propagators associated to the INF sentences we will present express that if some formula $\text{F}\{\overline{x} \mid L[\overline{x}]\} \leq y$ must be true and the assumption that $L[\overline{d}]$ is true (false) would imply that $\text{F}\{\overline{x} \mid L[\overline{x}]\}$ is certainly larger than y, then $L[\overline{d}]$ must be false (true). Similarly for formulas of the form $\text{F}\{\overline{x} \mid L[\overline{x}]\} \geq y$.

Let V be a set expression, I a structure and θ a variable assignment. We first investigate how to express that the value of an aggregate $\text{F}(I\theta(V))$ certainly increases or decreases if a tuple $\overline{d}$ is removed from $I\theta(V)$. If V is the set expression $\{(x_1, \ldots, x_n) \mid \varphi\}$, then we denote by $V \setminus (y_1, \ldots, y_n)$ the set expression $\{(x_1, \ldots, x_n) \mid (x_1 \neq y_1 \vee \ldots \vee x_n \neq y_n) \wedge \varphi\}$. Hence, if $I\theta(V) = \{\overline{d}_1, \overline{d}_2, \ldots, \overline{d}_m\}$, and $\theta(\overline{y}) = \overline{d}_1$ for some structure I and variable assignment θ, then $I\theta(V \setminus \overline{y}) = \{\overline{d}_2, \ldots, \overline{d}_m\}$. In other words, $V \setminus \overline{y}$ denotes the result of removing the tuple corresponding to $\overline{y}$ from V.

Table 4.2 defines the formulas $\text{cs}(\varphi)$ and $\text{cl}(\varphi)$ for atoms φ of the form $\text{F}(V \setminus \overline{y}) \leq z$ and $\text{F}(V \setminus \overline{y}) \geq z$. Intuitively, $\text{cs}(\varphi)$ expresses a condition that ensures the value of $\text{F}(V)$ is smaller than the value of $\text{F}(V \setminus \overline{y})$. Vice versa, $\text{F}(V)$ is larger than $\text{F}(V \setminus \overline{y})$ if $\text{cl}(\varphi) \wedge \varphi$ is satisfied. Formally, we have the following lemma.

Lemma 4.20. *Let I be a structure, θ a variable assignment, V the set expression $\{\overline{x} \mid \psi\}$ and φ a formula of the form $(\text{F}(V \setminus \overline{y}) \leq z)$ or $(\text{F}(V \setminus \overline{y}) \geq z)$. Then*

$$I\theta \models (\varphi \wedge \psi[\overline{x}/\overline{y}] \wedge \text{cs}(\varphi)) \Rightarrow (\text{F}(V) < \text{F}(V \setminus \overline{y})) \tag{4.34}$$

and

$$I\theta \models (\varphi \wedge \psi[\overline{x}/\overline{y}] \wedge \mathrm{cl}(\varphi)) \Rightarrow (\mathrm{F}(V) > \mathrm{F}(V \setminus \overline{y})). \tag{4.35}$$

Note that (4.34) and (4.35) are trivially satisfied in I if, respectively, $\mathrm{cs}(\varphi) = \bot$ and $\mathrm{cl}(\varphi) = \bot$.

Proof. The proof consists of a simple analysis of all cases. As an example, we prove (4.34) where φ is the formula $\mathrm{SUM}(\{(n, \overline{x}) \mid \psi\} \setminus (n', \overline{y})) \leq z$. Let I be a structure and θ a variable assignment such that $I\theta \models (\varphi \wedge \psi[\overline{x}/\overline{y}][n/n'] \wedge \mathrm{cs}(\varphi))$. It follows that $\theta(n') < 0$. Because $I\theta \models \psi[\overline{x}/\overline{y}][n/n']$ we have that

$$I\theta(\mathrm{SUM}(V)) = I\theta(\mathrm{SUM}(V \setminus (n', \overline{y}))) + \theta(n') < I\theta(\mathrm{SUM}(V \setminus (n', \overline{y}))).$$

■

The following definition extends the definition of $\mathrm{INF}(\varphi)$ to the case where φ is an ENF sentences containing an aggregate expression.

Definition 4.13. Let φ be an ENF sentence of the form $\forall \overline{x} \forall z\ (P(\overline{x}, z) \Leftrightarrow \mathrm{F}(V) \leq z)$, where V is the set expression $\{\overline{y} \mid L[\overline{x}, \overline{y}]\}$. Then $\mathrm{INF}(\varphi)$ is the set of INF sentences

$$\forall \overline{x} \forall z\ (\mathrm{F}(V) \leq z \Rightarrow P(\overline{x}, z)) \tag{4.36}$$
$$\forall \overline{x} \forall z\ (\mathrm{F}(V) > z \Rightarrow \neg P(\overline{x}, z)) \tag{4.37}$$
$$\forall \overline{x} \forall z \forall \overline{y}'\ (P(\overline{x}, z) \wedge \mathrm{F}(V \setminus \overline{y}') > z \Rightarrow L[\overline{x}, \overline{y}']) \tag{4.38}$$
$$\forall \overline{x} \forall z \forall \overline{y}'\ (\neg P(\overline{x}, z) \wedge \mathrm{F}(V \setminus \overline{y}') \leq z \Rightarrow L[\overline{x}, \overline{y}']) \tag{4.39}$$
$$\forall \overline{x} \forall z \forall \overline{y}'\ (P(\overline{x}, z) \wedge \mathrm{F}(V \setminus \overline{y}') \geq z \wedge \mathrm{cl}(\mathrm{F}(V \setminus \overline{y}') \geq z) \Rightarrow \neg L[\overline{x}, \overline{y}']) \tag{4.40}$$
$$\forall \overline{x} \forall z \forall \overline{y}'\ (\neg P(\overline{x}, z) \wedge \mathrm{F}(V \setminus \overline{y}') \leq z + 1 \wedge \mathrm{cs}(\mathrm{F}(V \setminus \overline{y}') \leq z + 1) \Rightarrow \neg L[\overline{x}, \overline{y}']). \tag{4.41}$$

If φ is of the form $\forall \overline{x} \forall z\ (P(\overline{x}, z) \Leftrightarrow \mathrm{F}(V) \geq z)$, where V is the set expression $\{\overline{y} \mid L[\overline{x}, \overline{y}]\}$, then $\mathrm{INF}(\varphi)$ is the set

$$\forall \overline{x} \forall z\ (\mathrm{F}(V) \geq z \Rightarrow P(\overline{x}, z)) \tag{4.42}$$
$$\forall \overline{x} \forall z\ (\mathrm{F}(V) < z \Rightarrow \neg P(\overline{x}, z)) \tag{4.43}$$
$$\forall \overline{x} \forall z \forall \overline{y}'\ (P(\overline{x}, z) \wedge \mathrm{F}(V \setminus \overline{y}') < z \Rightarrow L[\overline{x}, \overline{y}']) \tag{4.44}$$
$$\forall \overline{x} \forall z \forall \overline{y}'\ (\neg P(\overline{x}, z) \wedge \mathrm{F}(V \setminus \overline{y}') \geq z \Rightarrow L[\overline{x}, \overline{y}']) \tag{4.45}$$
$$\forall \overline{x} \forall z \forall \overline{y}'\ (P(\overline{x}, z) \wedge \mathrm{F}(V \setminus \overline{y}') \leq z \wedge \mathrm{cs}(\mathrm{F}(V \setminus \overline{y}') \leq z) \Rightarrow \neg L[\overline{x}, \overline{y}']) \tag{4.46}$$
$$\forall \overline{x} \forall z \forall \overline{y}'\ (\neg P(\overline{x}, z) \wedge \mathrm{F}(V \setminus \overline{y}') \geq z - 1 \wedge \mathrm{cl}(\mathrm{F}(V \setminus \overline{y}') \geq z - 1) \Rightarrow \neg L[\overline{x}, \overline{y}']). \tag{4.47}$$

Each of the sentences in $\mathrm{INF}(\varphi)$ is implied by φ. For sentences (4.36), (4.37), (4.42) and (4.43) this is straightforward. The fact that, e.g., (4.40) is implied by $\forall \overline{x} \forall z\ (P(\overline{x}, z) \Leftrightarrow \mathrm{F}(V) \leq z)$ is to be expected from its declarative reading: (4.40) states that if $P(\overline{x}, z)$ is true, the value of $\mathrm{F}(V)$ is larger than z if $\overline{y}'$ is left out, and adding $\overline{y}'$ would increase that value, i.e., make it strictly larger than z, then

$\overline{y}'$ should certainly be excluded from $\text{F}(V)$ in order to satisfy $\forall\overline{x}\forall z\ (P(\overline{x},z) \Leftrightarrow \text{F}(V) \leq z)$. The other sentences of $\text{INF}(\varphi)$ have similar intuitions. Since $\text{INF}(\varphi)$ clearly implies φ for every ENF sentence φ, we obtain the following proposition.

Proposition 4.21. $\text{INF}(\varphi)$ *is equivalent to* φ *for every ENF sentence* φ*.*

Proof. We prove the case where φ is of the form $\forall\overline{x}\forall z\ (P(\overline{x},z) \Leftrightarrow \text{F}(V) \leq z)$. Clearly, φ is equivalent to the conjunction of (4.36) and (4.37). Hence we only have to show that (4.38)–(4.41) are implied by φ. Let I be a structure that satisfies φ and let θ be a variable assignment. If $I\theta \models \neg L(\overline{x},\overline{y}')$, then $I\theta(V) = I\theta(V \setminus \overline{y}')$. It follows that if $I\theta \not\models L(\overline{x},\overline{y}')$ and $I\theta \models \text{F}(V \setminus \overline{y}') > z$, then $I\theta \models \text{F}(V) > z$ and therefore $I\theta \not\models P(\overline{x},z)$. We conclude that $\varphi \models$ (4.38). Similarly, it can be proven that $\varphi \models$ (4.39). To show that $\varphi \models$ (4.40), let I be a model of φ and θ a variable assignment. If $I\theta \models L[\overline{x},\overline{y}'] \wedge \text{F}(V \setminus \overline{y}') \geq z \wedge \text{cl}(\text{F}(V \setminus \overline{y}') \geq z)$, then Lemma 4.20 implies that $I\theta(\text{F}(V)) > I\theta(\text{F}(V \setminus \overline{y}')) \geq z$, and therefore $I\theta \not\models P(\overline{x},z)$. It follows that $\varphi \models$ (4.40). Similarly, it can be shown that $\varphi \models$ (4.41).
The case where φ is of the form $\forall\overline{x}\forall z\ (P(\overline{x},z) \Leftrightarrow \text{F}(V) \geq z)$ is analogous. ■

Definitions and symbolic propagation

To represent INF propagators using a positive definition and to define symbolic INF propagators, we used the fact that the value of a FO formula φ in a three-valued structure can be found by computing the value of the negation-free formulas φ^{ct} and φ^{cf}. We now extend the definition of φ^{ct} and φ^{cf} to FO(AGG) formulas, which then immediately lifts the results of Section 4.2.2 and Section 4.3 to FO(AGG).
To facilitate the definitions of φ^{ct} and φ^{cf}, we assume that $s^{\tilde{I}} \subset (\mathbb{N} \setminus \{0\})$ for every structure $\tilde{I}$ and sort $s \neq \mathfrak{Int}$ such that $\mathfrak{base}(s) = \mathfrak{Int}$. That is, the domains assigned by $\tilde{I}$ (except for $\mathfrak{Int}^{\tilde{I}}$) do contain neither zero nor negative numbers. See Appendix A for notes on the generalization to arbitrary finite structures.
Table 4.3 defines φ^{ct} in case φ is an atomic formula of the form $\text{F}(V) \leq y$ or $\text{F}(V) \geq y$. The correctness of, for instance, $(\text{SUM}\{(n,\overline{x}) \mid \psi\} \leq y)^{\text{ct}}$ can be seen as follows. In a finite structure $\tilde{I}$ where the domain of $\mathfrak{s}(n)$ only contains strictly positive values, the maximal value of $\text{SUM}\{(n,\overline{x}) \mid \psi\}$ in $\tilde{I}$ is given by $\sum_{(d_n,\overline{d}_x)\in V} d_n$, where $V = \{(d_n,\overline{d}_x) \mid \tilde{I}[n/d_n][\overline{x}/\overline{d}_x](\psi) \neq \mathbf{f}\}$. Clearly, this sum is equal to

$$\left(\sum_{(d_n,\overline{d}_x)\in\mathfrak{s}((n,\overline{x}))^{\tilde{I}}} d_n\right) - \left(\sum_{(d_n,\overline{d}_x)\in V'} d_n\right),$$

where V' is the set $\{(d_n,\overline{d}_x) \mid \tilde{I}[n/d_n][\overline{x}/\overline{d}_x](\psi) = \mathbf{f}\}$. Hence, the maximal value $\text{SUM}\{(n,\overline{x}) \mid \psi\}$ is less than $\tilde{I}^{\text{tf}}(\text{SUM}\{(n,\overline{x}) \mid \top\} - (\text{SUM}\{(n,\overline{x}) \mid \psi^{\text{cf}}))$. The formula $(\text{SUM}\{(n,\overline{x}) \mid \psi\} \leq y)^{\text{ct}}$ in Table 4.3 expresses precisely that the maximal value is less than y.

φ	φ^{ct}
$\textsc{card}\{\overline{x} \mid \psi\} \geq y$	$\textsc{card}\{\overline{x} \mid \psi^{\mathrm{ct}}\} \geq y$
$\textsc{sum}\{(n,\overline{x}) \mid \psi\} \geq y$	$\textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{ct}}\} \geq y$
$\textsc{prod}\{(n,\overline{x}) \mid \psi\} \geq y$	$\textsc{prod}\{(n,\overline{x}) \mid \psi^{\mathrm{ct}}\} \geq y$
$\textsc{min}\{(n,\overline{x}) \mid \psi\} \geq y$	$\forall n \forall \overline{x}\ (\psi^{\mathrm{cf}} \vee n \geq y)$
$\textsc{max}\{(n,\overline{x}) \mid \psi\} \geq y$	$\exists n \exists \overline{x}\ (\psi^{\mathrm{ct}} \wedge n \geq y)$
$\textsc{card}\{\overline{x} \mid \psi\} \leq y$	$\exists y_1 \exists y_2\ (\textsc{card}\{\overline{x} \mid \psi^{\mathrm{cf}}\} \geq y_1 \wedge$ $\textsc{card}\{\overline{x} \mid \top\} \leq y_2 \wedge y_2 - y_1 \leq y)$
$\textsc{sum}\{(n,\overline{x}) \mid \psi\} \leq y$	$\exists y_1 \exists y_2\ (\textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{cf}}\} \geq y_1 \wedge$ $\textsc{sum}\{(n,\overline{x}) \mid \top\} \leq y_2 \wedge y_2 - y_1 \leq y)$
$\textsc{prod}\{(n,\overline{x}) \mid \psi\} \leq y$	$\exists y_1 \exists y_2\ (\textsc{prod}\{(n,\overline{x}) \mid \psi^{\mathrm{cf}}\} \geq y_1 \wedge$ $\textsc{prod}\{(n,\overline{x}) \mid \top\} \leq y_2 \wedge y_2/y_1 \leq y)$
$\textsc{min}\{(n,\overline{x}) \mid \psi\} \leq y$	$\exists n \exists \overline{x}\ (\psi^{\mathrm{ct}} \wedge n \leq y)$
$\textsc{max}\{(n,\overline{x}) \mid \psi\} \leq y$	$\forall n \forall \overline{x}\ (\psi^{\mathrm{cf}} \vee n \leq y)$

Table 4.3: Extending φ^{ct} to FO(AGG)

We define $(\textsc{f}(V) \leq y)^{\mathrm{cf}}$ by $(\exists z\ (\textsc{f}(V) \geq z)^{\mathrm{ct}} \wedge z > y)$ and $(\textsc{f}(V) \geq y)^{\mathrm{cf}}$ by $(\exists z\ (\textsc{f}(V) \leq z)^{\mathrm{ct}} \wedge z < y)$. Furthermore, $(\textsc{f}(V) = y)^{\mathrm{ct}}$ is defined by $((\textsc{f}V \leq y)^{\mathrm{ct}} \wedge (\textsc{f}V \geq y)^{\mathrm{ct}})$ and $(\textsc{f}(V) = y)^{\mathrm{cf}}$ by $((\textsc{f}V \leq y)^{\mathrm{cf}} \wedge (\textsc{f}V \geq y)^{\mathrm{cf}})$.
The following proposition extends Proposition 2.7 to FO(AGG).

Proposition 4.22. *Let $\tilde{I}$ be a finite structure such that $d > 0$ for every $d \in s^{\tilde{I}}$, where s is a subsort of $\mathfrak{Int}$. Let θ be a variable assignment and φ a* FO(AGG) *formula in TNF. Then $\tilde{I}\theta(\varphi) = \tilde{I}^{\mathrm{tf}}\theta(\varphi^{\mathrm{ct}}, \varphi^{\mathrm{cf}})$.*

Proof. See Appendix A ■

Since we assumed that domains do not contain negative numbers, all formulas φ^{ct} in Table 4.3 are monotone. From the definition of φ^{ct} and φ^{cf} for arbitrary TNF formulas φ, it then follows that both φ^{ct} and φ^{cf} are monotone. Hence Δ_V is a monotone definition for every set V of INF propagators.
Now the results of Section 4.2.2 and Section 4.3 extend to FO(AGG). In particular, Proposition 4.10 and Proposition 4.16 hold for INF propagators containing aggregates. The proofs remain the same since they only depend on Proposition 2.7 and on the fact that Δ_V is monotone for every set V of INF propagators.

4.4.4 Definitions

In this section, we consider two approaches to extend the propagation method to FO(ID) and FO($\cdot$). The first one simply applies the propagation method for

FO(AGG) theories on the completion of FO($\cdot$) theories. The second one defines an INF propagator for definitions. To guarantee that propagation produces finite structures, we again assume that $\mathfrak{Int} \notin \mathfrak{s}(P)$ and $\mathfrak{Int} \notin \mathfrak{s}(F)$ for every predicate symbol P and function symbol F that is not "built-in".

Propagation on the completion

A simple way to extend the propagation method we developed so far to FO($\cdot$) relies on the fact that any propagator O for the completion Comp(T) of a FO($\cdot$) theory T is also a propagator for T itself. Indeed, if $\tilde{I}$ is a structure, M a model of T such that $\tilde{I} \leq_p M$, then Theorem 3.6 ensures that $M \models \text{Comp}(T)$, and hence $O(\tilde{I}) \leq_p M$. We conclude that applying the propagation method we developed so far, i.e., Algorithm 4.2 for FO(AGG) theories, on Comp(T) is a propagation method for T. The benefits of this approach that the results we obtained so far, for instance, representing the propagation by a monotone definition and symbolic propagation, carry over to FO($\cdot$). On the other hand, the propagation method is clearly incomplete. For example, if a domain atom $P(\overline{d})$ is unknown in $\tilde{I}$, false in every model of T approximated by $\tilde{I}$, but true in some model of Comp(T), then $\mathscr{O}^T(\tilde{I})(P(\overline{d})) = \mathbf{f}$, while $O(\tilde{I})(P(\overline{d})) = \mathbf{u}$ for any propagator O for Comp(T).

Propagators for definitions

A second way to extend our propagation method to FO(ID) and FO($\cdot$) is by introducing INF[20] propagators involving definitions.

Definition 4.14. The propagator $\mathscr{I}^\Delta$ for a definition Δ is defined by

$$P^{\mathscr{I}^\Delta(\tilde{I})}(\overline{d}) = \begin{cases} \text{lub}_{\leq_p}\{\mathbf{t}, P^{\tilde{I}}\} & \text{if } P^{\text{wfm}_\Delta(\tilde{I})}(\overline{d}) = \mathbf{t} \\ \text{lub}_{\leq_p}\{\mathbf{f}, P^{\tilde{I}}\} & \text{if } P^{\text{wfm}_\Delta(\tilde{I})}(\overline{d}) = \mathbf{f} \\ P^{\tilde{I}} & \text{otherwise.} \end{cases}$$

It follows from the definition of well-founded induction that $\mathscr{I}^\Delta$ is a monotone propagator for every definition Δ. For the class of finite structures $\tilde{I}$ we consider in this section, all well-founded inductions are finite sequences. In fact, $\text{wfm}_\Delta(\tilde{I})$ can be computed in polynomial time in $|\tilde{I}|$. As such, $\mathscr{I}^\Delta$ is a propagator with polynomial time data-complexity.

It is still an open question whether the propagator $\lim_{\mathscr{I}(V)}$ can be represented by a (positive or monotone) definition if V may contain both INF sentences and definitions. Results from fixpoint logics (see, e.g., Grädel et al., 2007) suggest that this will be possible when only finite structures are considered, but impossible in general. We expect that even if it is possible to represent $\lim_{\mathscr{I}(V)}$ by a definition Δ_V, this result will not be of practical importance, since Δ_V

[20] We call the propagators for definitions *INF* propagators only because all other polynomial time propagators we introduced so far were INF propagators. Yet, there does not exist INF sentences involving definitions, since definitions cannot be strict subformulas of sentences.

will be rather complicated. The same remark applies for symbolic propagators simulating $\lim_{\mathscr{I}(V)}$.

4.5 Applications

In this section, we discuss three applications of constraint propagation, namely finite model generation, declarative programming of configuration systems and approximate query answering in incomplete databases. Two other applications are explained in Chapter 5 and Chapter 6.

4.5.1 Finite model generation

Model generation is the problem of computing a model of a logic theory T, usually in the context of a given finite domain, typically the Herbrand universe. A model generator can decide the satisfiability of the theory in the context of this fixed domain. This is useful, e.g., in the context of lightweight verification (Jackson, 2006). Beyond determining satisfiability, there is a broad class of problems of which the answers are naturally given by the models of a logic theory. For example, the model of a theory specifying a scheduling domain typically contains a (correct) schedule. Thus, a model generator applied to this theory will solve the scheduling problem for this domain.[21] This idea of model generation as a declarative problem solving paradigm has been pioneered in the area of Answer Set Programming (ASP) (Marek and Truszczyński, 1999; Niemelä, 1999). In this area, answers to a problem are given by the models of a normal logic program under the stable model semantics (Gelfond and Lifschitz, 1988). Earlier, SAT solvers had been used in this spirit, for example in Kautz and Selman's blackbox approach to planning problems (Kautz and Selman, 1996). And, as recently pointed out by Mitchell and Ternovska (2008), problem solving in Constraint Programming (CP) systems often amounts to computing models of first-order logic (FO) specifications.

Many practical model generation problems contain additional data besides the input theory and finite domain. This data can be implicit in the input theory. For example, ASP problems can be split into two parts: a non-ground theory and a list of ground facts. The latter part essentially represents given data. In other contexts (Mitchell and Ternovska, 2005; Torlak and Jackson, 2007; Wittocx et al., 2008d), the data is given as a (three-valued) structure interpreting part of the vocabulary of the input theory. In the following, we assume without loss of generality that the data is represented by a structure. In practice, it is often the case that some preprocessing, e.g., materializing a view on a database, needs to be done before the data is in this format.

Model generation with an input theory and input structure is called *model expansion*. Model expansion for a logic $\mathcal{L}$, denoted MX($\mathcal{L}$), is defined as follows.

[21] For a set of problems of this kind, see, e.g., the benchmarks of the ASP-competition (`http://www.cs.kuleuven.be/~dtai/events/ASP-competition`).

Definition 4.15. Let T be an $\mathcal{L}$-theory over a vocabulary Σ, σ a subvocabulary of Σ and I_σ a finite σ-structure. The *model expansion search problem with input* $\langle T, I_\sigma \rangle$ is the problem of computing a Σ-structure M such that $M \models T$ and $M|_\sigma = I_\sigma$.

The vocabulary σ is called the *input vocabulary* of the problem, the vocabulary $\Sigma \setminus \sigma$ the *expansion vocabulary*. I_σ is called the *input structure*. Because we required that σ and Σ have exactly the same sorts (see the definition of *subvocabulary* in Section 2.1.1), it follows that I_σ completely determines the domain of the solutions to the model expansion search problem with input $\langle T, I_\sigma \rangle$.
Observe that if $\sigma = \Sigma$, model expansion reduces to model checking, while if $\sigma = \langle \emptyset, \emptyset \rangle$, it reduces to model generation for T with a given finite domain size. Also, if T is a theory over a vocabulary Σ containing no function symbols of arity greater than zero, Herbrand model generation for T can be simulated by model expansion. Indeed, let $\sigma = \langle \Sigma_S, \emptyset, \Sigma_F \rangle$, and I_σ the structure with the Herbrand universe of T such that $C^{I_\sigma} = C$ for every constant $C \in \Sigma_F$.
We illustrate model expansion by two examples.

Example 4.22 (Graph Colouring)**.** The graph colouring problem is the problem of colouring a given graph with a given set of colours such that adjacent vertices have different colours. To express this problem in MX(FO), let *Vtx* and *Col* be sorts and let σ contain the predicate $Edge(Vtx, Vtx)$. The sort *Col* denotes the given set of colours, the given graph is represented by *Vtx* and *Edge*. Let Σ be the vocabulary that contains besides *Edge* also the function symbol $Colour(Vtx) : Col$ and let T be the theory that consists of the sentence

$$\forall v_1 \forall v_2 \ (Edge(v_1, v_2) \Rightarrow Colour(v_1) \neq Colour(v_2)).$$

Then model expansion with input theory T and input vocabulary σ expresses the graph colouring problem. Indeed, for any $M \models T$ that expands I_σ, $Colour^M$ is a proper colouring of the graph represented by I_σ.

Example 4.23 (SAT)**.** To represent the SAT problem for propositional CNF theories in MX(FO), let σ be a vocabulary containing the two sorts *Atom* and *Clause*, representing the atoms and the clause of the input CNF theory, and the two predicates $PosIn(Atom, Clause)$ and $NegIn(Atom, Clause)$, to represent the positive, respectively negative, occurrences of atoms in clauses. The theory given by

$$\forall c \ \exists a \ ((PosIn(a, c) \wedge True(a)) \vee (NegIn(a, c) \wedge \neg True(a)))$$

over a $\sigma \cup \{True(Atom)\}$ expresses the SAT problem: for any $M \models T$ that expands I_σ, the propositional structure represented by $True^M$ is a model of the CNF theory represented by I_σ. Indeed, the theory forces that every clause contains at least one true literal.

It has to be noted that there are model generation problems where (part of) the data is not naturally represented by a finite structure. An example is the

battleship puzzle shown in Figure 2.1. In Chapter 2 we showed that the given hints of this puzzle can be described by a finite structure, but this required encoding the fact that square $(2, 3)$ contains "the first part of a ship of at least length two which is heading north" by stating that $(1, 3)$ does not contain a ship while both $(2, 3)$ and $(3, 3)$ do. It is more natural to represent this data by the sentence

$$\begin{aligned}\exists s\ (&\mathit{InitRow}(s) = 2 \wedge \mathit{InitCol}(s) = 3\\ &\wedge \mathit{ShipLength}(s) \geq 2 \wedge \mathit{ShipDir}(s) = \mathit{Vertical}).\end{aligned}$$

As shown by Mitchell and Ternovska (2005), it follows from Fagin's (1974) theorem that model expansion for FO *captures* **NP**, in the following sense:

- For any fixed T and σ the problem of deciding whether there exists a model of T expanding an input structure I_σ is in **NP**.
- Vice versa, for any **NP** decision problem X on the class of finite σ-structures there is a vocabulary $\Sigma \supseteq \sigma$ and a first-order Σ-theory T such that model expansion with input theory T *expresses* X, i.e., I_σ belongs to X iff there exists a Σ-structure M expanding I_σ such that $M \models T$.

This result proves that any **NP** problem X can be expressed by an MX(FO) problem, and hence shows the broad applicability of MX(FO) solvers to solve **NP** problems. Ternovska and Mitchell (2009) obtained a similar result for finite structures containing integer arithmetic.
In the rest of this section, we use the following slightly more general form of model expansion.

Definition 4.16. Let T be a theory over Σ and $\tilde{I}$ a finite four-valued Σ-structure. The model expansion search problem with input $\langle T, \tilde{I}\rangle$ is the problem of computing a two-valued Σ-structure M such that $\tilde{I} \leq_p M$ and $M \models T$.

If $\tilde{I}$ is two-valued on a vocabulary $\sigma \subseteq \Sigma$ and $\tilde{I}|_{\Sigma\setminus\sigma} = \bot^{\leq_p}$, then the model expansion search problem with input $\langle T, \tilde{I}\rangle$ reduces to a model expansion as defined in Definition 4.15 with $\tilde{I}|_\sigma$ as input structure.

A simple finite model expander

Algorithm 4.4 presents a simple backtracking algorithm, based on constraint propagation, to solve the MX search problem with input $\langle T, \tilde{I}\rangle$. As a first step, constraint propagation for T is applied on $\tilde{I}$ by the function `propagate`. This step can be implemented by Algorithm 4.2. If the resulting structure is two-valued, this structure is returned, if it is strictly four-valued, the most precise structure $\top^{\leq_p}$ is returned, indicating that there is no model of T approximated by $\tilde{I}$. If the resulting structure is strictly three-valued, a domain atom $P(\overline{d})$ such that $P^{\tilde{I}} = \mathbf{u}$ is chosen by the function `choose`. Then, a model of T approximated by $\tilde{I}[P(\overline{d})/\mathbf{t}]$ is searched for by calling `Expand` on T and $\tilde{I}[P(\overline{d})/\mathbf{t}]$. If such a

Algorithm 4.4: Expand

Input: A theory T and finite structure $\tilde{I}$

```
1 Ĩ := propagate(T,Ĩ);
2 if Ĩ is two-valued then return Ĩ;
3 else if Ĩ is strictly four-valued then return ⊤^≤p;
4 else
5     P(d̄) := choose(Ĩ);
6     J̃ := Expand(T,Ĩ[P(d̄)/t]);
7     if J̃ is two-valued then return J̃;
8     else return Expand(T,Ĩ[P(d̄)/f]);
```

model does not exist, the algorithm tries to find a model for T approximated by $\tilde{I}[P(\overline{d})/\mathbf{f}]$.

A propagator O for T *induces* T if $O(I)$ is strictly four-valued for any two-valued structure $I \not\models T$. If `propagate` is implemented by a propagator that induces T, `Expand` correctly implements finite model generation.

Proposition 4.23. *If* `propagate` *is implemented by a propagator that induces* T*, then* `Expand`$(T, \tilde{I})$ *returns a model* M *of* T *such that* $\tilde{I} \leq_p M$ *if such a structure* M *exists. Otherwise it returns* $\top^{\leq_p}$*.*

If terminal refinement sequences are in step 3 of Algorithm 4.2, then this algorithm implements a propagator that induces T. Indeed, let I be a two-valued structure that does not satisfy T and let $\mathrm{INF}(T)$ be the set of INF sentences associated to T as in Algorithm 4.2. Since $\mathrm{INF}(T)$ is equivalent to T, there exists a sentence $\varphi \in \mathrm{INF}(T)$ such that $I \not\models \varphi$. The sentence φ is of the form $\forall \overline{x}\ (\psi \Rightarrow L[\overline{x}])$. Hence there exists a tuple $\overline{d}$ such that $I[\overline{x}/\overline{d}] \models \psi$ and $I[\overline{x}/\overline{d}](L[\overline{x}]) = \mathbf{f}$. It follows that $\mathscr{I}^{\varphi}(I)[\overline{x}/\overline{d}](L[\overline{x}]) = \mathbf{i}$. Therefore, a terminal $\mathscr{I}(\mathrm{INF}(T))$-refinement sequence from I is strictly four-valued, which proves that Algorithm 4.2 implements a propagator that induces T if it constructs terminal refinement sequences.

State-of-the-art finite model generators do not implement an algorithm similar to Algorithm 4.4. Instead, they use *grounding* to reduce their input to an equivalent SAT problem and then call an efficient SAT solver. In the next chapter, we investigate how to improve grounding size and speed by applying (first-order) constraint propagation.

A naive implementation of Algorithm 4.4 will be far slower than current state-of-the-art model generators on most applications. Yet, as noted by Mellarkod et al. (2008), first-order constraint propagation for at least part of the input theory is often necessary to avoid impractically large groundings. In Chapter 6, we illustrate another benefit of Algorithm 4.4, namely that it yields a solver that is suitable for debugging FO theories by tracing. In that context, the efficiency of the solver is often less important.

Constraint satisfaction problems

Finite model generation problems are closely related to *constraint satisfaction problems* (CSP). A CSP consists of a finite set V of variables, for each variable $v \in V$ a finite set $\mathrm{dom}(v)$ of possible values, called the domain of v, and a set of constraints $\mathcal{C}$ over the variables. Each constraint over variables $v_1, \ldots, v_n$ is a subset of $\mathrm{dom}(v_1) \times \cdots \times \mathrm{dom}(v_n)$. A solution for a CSP problem $\langle V, \mathcal{C} \rangle$ is a function F, mapping each variable v to a value $F(v)$ in its domain such that for each constraint $C \in \mathcal{C}$ over the variables $(v_1, \ldots, v_n)$, $(F(v_1), \ldots, F(v_n)) \in C$.
A system that solves CSPs provides the user with a language to express several useful constraints. E.g., '$v_1 + v_2 \leq 4$' can be used to denote the constraint $\{(d_1, d_2) \in \mathrm{dom}(v_1) \times \mathrm{dom}(v_2) \mid d_1 + d_2 \leq 4\}$ and '`AllDifferent`$(v_1, \ldots, v_n)$' to denote the constraint

$$\{(d_1, \ldots, d_n) \in \mathrm{dom}(v_1) \times \cdots \times \mathrm{dom}(v_n) \mid d_i \neq d_j \text{ for each } i \neq j\}.$$

Current CSP solvers implement variants of algorithm `Expand`. Their efficiency stems from the fact that they implement efficient special purpose propagators for the constraints, e.g., an efficient propagator for the `AllDifferent` constraint, and good search strategies, i.e., good heuristics behind the implementation of `choose`. Some CSP solvers allow the user to easily specify different search strategies. COMET (Michel and Van Hentenryck, 2005) is an example of such a solver.
A CSP $\langle V, \mathcal{C} \rangle$ can easily be transformed to an MX problem $\langle T, \tilde{I} \rangle$, by introducing a constant C_v of sort s for each variable $v \in V$ and assigning $s^{\tilde{I}} = \mathrm{dom}(v)$. For each constraint $C \in \mathcal{C}$ over $v_1, \ldots, v_n$, the theory T should contain an atomic sentence $P(C_{v_1}, \ldots, C_{v_n})$, where $P^{\tilde{I}}$ is two-valued and equal to C. Finally, $\tilde{I}$ assigns $(\emptyset, \emptyset)$ to each constant C_v, i.e., the value of C_v is unknown in $\tilde{I}$. If M is a solution to the model expansion problem with input $\langle T, \tilde{I} \rangle$, then $F : v \mapsto C_v^M$ is a solution to $\langle V, \mathcal{C} \rangle$. Vice versa, for each solution F of the CSP, the structure $M \geq_p \tilde{I}$ obtained by assigning $C_v^M = F(v)$ is a model of T.
Since finite model generation and CSP are so closely related, it is an interesting topic for future research to investigate how the efficient propagation techniques and heuristics from CSP solvers can be applied to improve the presented basic model generator (Algorithm 4.4). One difficulty is that, unlike the translation from a CSP to a model generation problem, the inverse translation is less straightforward. For example, in a modelling of a Sudoku puzzle, the constraint that a row may not contain the same number more than once could be expressed by

$$\forall r \forall c_1 \forall c_2 \ (c_1 \neq c_2 \Rightarrow \mathit{Sudoku}(r, c_1) \neq \mathit{Sudoku}(r, c_2)), \tag{4.48}$$

where *Sudoku* represents a function that maps every position of the grid to the number it contains. Typically, the same rule is expressed in a CSP by the constraint

$$\texttt{AllDifferent}(v_{r,1}, \ldots, v_{r,9}), \tag{4.49}$$

where each variable $v_{r,i}$ denotes the number at position (r, i). To be able to apply an efficient `AllDifferent` propagator for propagation on (4.48), a system

should be able to automate the translation to (4.49). That is, it should be able to recognize that (4.48) is in fact an `AllDifferent` constraint.
Approaches for integrating CSP solvers in model generators that rely on grounding were investigated by Mellarkod et al. (2008) and Gebser et al. (2009b).

4.5.2 Configuration systems

The application presented in the introduction to this chapter is an example of a *configuration system*. A configuration system helps a user to fill out a form in accordance with certain constraints. As noted by Vlaeminck et al. (2009), due to the large amount of background knowledge involved, developing and maintaining a configuration system is typically much easier using a knowledge base system compared to using a traditional imperative programming method. One of the tasks of a configuration system is to prevent the user from making invalid choices by automatically disabling such choices. For example, if courses C_1 and C_2 cannot be followed both and a student selects the course C_1, the system described in the introduction should make selecting C_2 impossible. Using constraint propagation, this functionality can be implemented in a declarative way: the constraints describing valid configurations are represented by a theory T, the current selection by a three-valued structure $\tilde{I}$. Then, propagation is applied to derive a more precise structure $\tilde{J}$. Each possible choice that is true according to $\tilde{J}$ is selected automatically by the system, each choice that is false is disabled.
There are two main requirements for the propagation in this case. First, since a configuration system is interactive, the propagation should be efficient in order to respond sufficiently fast. Secondly, in an ideal system, the user can never make an invalid choice. To this end, the propagation should implement the complete propagator $\mathscr{O}^T$. Indeed, if $\tilde{J} = \mathscr{O}^T(\tilde{I})$ and a choice $P(\overline{d})$ is unknown in $\tilde{J}$, then there exists a model of T, i.e. a valid configuration, where $P(\overline{d})$ is true, and one where $P(\overline{d})$ is false. As such, neither selecting nor deselecting $P(\overline{d})$ is an invalid choice since in both cases a valid configuration remains reachable. The combination of both requirements shows the importance of investigating the precision of propagators.
We refer to the work of Vlaeminck et al. (2009) for a more elaborated investigation of knowledge based configuration software and a discussion of related work. As a proof of concept, they implemented a course selection application based on the propagation algorithm using symbolic INF propagators as presented in this chapter.[22] For this small example, our propagation method turns out to be sufficiently fast and precise.

4.5.3 Approximate query answering

A recent trend in databases is the development of approximate methods to reason about databases with incomplete knowledge. The incompleteness of the

[22] The application can be downloaded from `www.cs.kuleuven.be/~hanne/demo/`.

database may stem from the use of null values, or of a restricted form of closed world assumption (Cortés Calabuig et al., 2007), or it arises from integrating a collection of local databases each based on its own *local schema* into one virtual database over a *global schema* (Grahne and Mendelzon, 1999). In all these cases, the data complexity of certain and possible query answering is computationally hard (**coNP**, respectively **NP**). For this reason fast (and often very precise) polynomial approximate query answering methods have been developed, which compute an underestimation of the certain, and an overestimation of the possible answers.

The tables of an incomplete database are naturally represented as a three-valued structure $\tilde{I}$. The integrity constraints, local closed world assumption or mediator scheme corresponds to a logic theory T. Answering a query $\{\overline{x} \mid \varphi\}$ boils down to computing the set of tuples $\overline{d}$ such that $M[\overline{x}/\overline{d}] \models \varphi$ in every model M of T approximated by $\tilde{I}$ and the set of $\overline{d}$ such that $M[\overline{x}/\overline{d}] \models \varphi$ for at least one $M \models T$ approximated by $\tilde{I}$. These sets can be approximated by $\{\overline{d} \mid \tilde{J}[\overline{x}/\overline{d}](\varphi) = \mathbf{t}\}$, respectively $\{\overline{d} \mid \tilde{J}[\overline{x}/\overline{d}](\varphi) \neq \mathbf{f}\}$, where $\tilde{J}$ is obtained by applying constraint propagation for T on $\tilde{I}$. If a constraint propagation method with polynomial data-complexity is used to compute $\tilde{J}$, computing the approximate query answers above also requires polynomial time in the size of the database. Of course, the more precise $\tilde{J}$ is, the more precise the obtained answers to the query are.

Approximate query answering is an application where symbolic propagation is important. There are several reasons why it is to be preferred above non-symbolic propagation. First of all, the size of real-life databases makes the application of non-symbolic propagation often too slow in practice, since it requires the storage of large intermediate tables. More importantly, each time the data is changed, the propagation needs to be repeated. This is not the case for the symbolic propagation, because symbolic propagation is independent of the data. Thirdly, symbolic propagation can be used for query rewriting. Indeed, given a symbolic structure $\tilde{\Phi}$, computed by propagation, an evaluation structure E and a query φ, the approximation to the certain answers for φ are given by the set $\{\overline{d} \mid E(\tilde{\Phi})[\overline{x}/\overline{d}](\varphi) = \mathbf{t}\}$. This set is equal to $\{\overline{x} \mid \varphi^{\tilde{\Phi}^{\mathrm{ct}}}\}^{E^{\mathrm{tf}}}$. This shows the query $\{\overline{x} \mid \varphi\}$ can be rewritten to a new query $\{\overline{x} \mid \varphi^{\tilde{\Phi}^{\mathrm{ct}}}\}$, which is then evaluated in the database E^{tf}. One can make use of the various optimization strategies in current database management systems to efficiently compute the answers to the new query. Possible answers to φ are obtained in a similar way. Applying the non-symbolic version of Algorithm 4.2 for approximate query answering generalizes the algorithm of Cortés Calabuig et al. (2006). Applying the symbolic version and rewriting the query generalizes the query rewriting technique presented by Cortés Calabuig et al. (2007).

4.6 Conclusion

In this chapter we presented constraint propagation as a basic form of inference for FO(·) theories. We introduced a general notion of constraint propagators

and briefly discussed the complete propagator for a theory. Due to its high computational complexity, the complete propagator cannot be applied in a real-life knowledge base system. Therefore we investigated incomplete propagators, called INF propagators. Besides their lower computational complexity, INF propagators for FO have other interesting properties: propagation using INF propagators can be represented by a monotone definition and can be executed in a symbolic manner. The former property allows us to use existing systems and extensively studied methods to make efficient implementations of propagation. The latter property is important in contexts where data changes regularly or where only part of the results obtained by propagation is needed.
We extended the results about propagation using INF propagators to full FO($\cdot$). All important properties of INF propagators are preserved when adding aggregates. Whether the results about representation by a monotone definition or symbolic propagation carry over to FO(ID), is an open question.
Finally, we discussed several applications that rely on constraint propagation as basic form of inference. These applications, and the ones that will be presented in the next chapters, indicate the importance of constraint propagation in the context of a knowledge base system.

Chapter 5

Grounding

Grounding, or propositionalization, is the task of reducing a first-order theory and finite domain to an equivalent propositional theory, called *a grounding*. Grounding is used as a preprocessing phase in many logic-based reasoning systems. It serves to provide the user with a rich input language, while enabling the system to rely on efficient propositional solvers to perform the actual reasoning. Examples of systems that rely on grounding can be found in the area of finite first-order model generation (Claessen and Sörensson, 2003; McCune, 2003; East et al., 2006; Mitchell et al., 2006; Torlak and Jackson, 2007; Wittocx et al., 2008d). Such systems are used as part of theorem provers (Claessen and Sörensson, 2003) and for lightweight software verification (Jackson, 2006). Currently, almost all Answer Set Programming (ASP) systems rely on grounding as a preprocessing phase (Gebser et al., 2007; Perri et al., 2007; Syrjänen, 2000; Syrjänen, 2009). Also in planning systems (Kautz and Selman, 1996) and relational data mining (Krogel et al., 2003) grounding is frequently used. This large number of applications indicates the importance of grounding in logic-based reasoning systems and the need to develop efficient grounders.

A basic (naive) grounding method is by instantiating the variables in the input theory by all possible combinations of domain elements. Grounding in this way is polynomial in the size of the domain but exponential in the maximum width of a formula in the input theory, and may easily produce groundings of unwieldy size. Several techniques have been developed to efficiently produce smaller groundings. There are two main categories of such techniques. In the first, the input theory is rewritten such that the maximum width of the formulas decreases. Methods like *clause splitting* (Schulz, 2002) and *partitioning* (Ramachandran and Amir, 2005) belong to this category.

The second type of techniques is applicable when besides the finite domain, additional data is available. This is often the case in practical model generation problems, such as the ones that are typical in ASP. In a graph problem the data could be an encoding of the input graph; in the context of planning, it could be a description of the initial and goal state, etc. Sometimes the data is explicitly available, e.g., in the form of a database, sometimes it is implicit, e.g., as a

set of ground facts in the input theory. The second type of techniques aims at efficiently computing small groundings by taking the data into account. Both types of techniques can be combined in a grounder.
We mainly focus on a technique of the second category. To explain the intuition underlying our method, consider the following model generation problem.

Example 5.1. Let T_1 be the first-order logic theory over the vocabulary $\{Edge, Sub\}$, consisting of the two sentences

$$\forall u \forall v \ (Sub(u, v) \Rightarrow Edge(u, v)), \tag{5.1}$$

$$\forall x \forall y \forall z \ (Sub(x, y) \wedge Sub(x, z) \Rightarrow y = z). \tag{5.2}$$

T_1 expresses that Sub is a subgraph of $Edge$ with at most one outgoing edge in each vertex. Computing such a subgraph of a given graph $G = \langle V, E \rangle$ can be cast as a model generation problem with input theory T_1 and data G. The data can be represented as a structure I_σ for the subvocabulary $\sigma_1 = \{Edge\}$ with domain V and $Edge^{I_\sigma} = E$. A solution is obtained by generating a model of T_1 that expands I_σ with an interpretation of Sub.
Applying the naive grounding algorithm produces $|V|^2$ instantiations of (5.1) and $|V|^3$ instantiations of (5.2). By taking the data into account, atoms over $Edge$ and '=' can be substituted by their truth value in I_σ. Simplifying the resulting grounding then eliminates $|E|$ instantiations of (5.1) and $|V|$ instantiations of (5.2). Smart grounding algorithms interleave this substitution and simplification with the grounding process in order to avoid creating unnecessary parts of the grounding.

Observe that substituting atoms over σ_1 and then simplifying still produces a grounding of size $\mathcal{O}(|V|^3)$. Indeed, the simplified grounding of (5.2) is the set of binary clauses $\neg Sub(i, j) \vee \neg Sub(i, k)$ such that $i, j, k \in V$ and $i \neq j$. This set has size $|V|^3 - |V|$.
Some grounders apply reasoning on the ground theory to reduce it even further. In the example, the simplified grounding of (5.1) consists of the clauses $\neg Sub(i, j)$ such that $(i, j) \notin E$. Since these clauses contain only one literal, each of these literals certainly true in every model of the ground theory. It follows that each binary clauses $\neg Sub(i, j) \vee \neg Sub(i, k)$ such that either $\neg Sub(i, j)$ or $\neg Sub(i, k)$ belongs to the simplified grounding of (5.1) is certainly true in every model of the ground theory and thus can be omitted from the simplified grounding of (5.2). The result is a grounding of size $|E \bowtie_{1=1} E|$, where $\bowtie_{1=1}$ denotes the natural join matching the first columns. For a sparse graph, $|E \bowtie_{1=1} E|$ is much smaller than $|V|^3$. However, since reasoning on the ground theory does not avoid creating all instantiations of a formula, it does not significantly speed up the grounding process.
One way to avoid a large grounding without relying on reasoning on the ground theory is by adding redundant information to formulas. This method is frequently used in ASP. For example,

$$\forall x \forall y \forall z \ (Edge(x, y) \wedge Sub(x, y) \wedge Edge(x, z) \wedge Sub(x, z) \Rightarrow y = z) \tag{5.3}$$

is equivalent to (5.2) given (5.1), but its grounding (without reasoning on the ground theory) is equal to the one obtained by the kind of reasoning on the ground theory illustrated above. This illustrates how adding redundant information may sometimes dramatically reduce the size of the grounding. Since current grounders are optimized to ground formulas like (5.3) without actually trying all redundant instances, grounding may also speed up a lot.
However, manually adding redundancy to formulas has its disadvantages: it leads to more complex and hence, less readable theories. Worse, it might introduce errors. It requires a good understanding of the used grounder, since it depends on the grounder what information is beneficial to add and where. Also, a human developer could easily miss useful information.
The above motivates a study of automated methods for deriving such redundant information and of principled ways of adding it to formulas. We show that symbolic constraint propagation on the input theory T can be used to derive such redundant information, in the form of a pair of a *symbolic upper and lower bound* for each subformula of T. Each of these bounds is a formula over the vocabulary of the input structure I_σ. For instance, for Example 5.1, our algorithm will compute $Edge(x, y)$ as upper bound for $Sub(x, y)$, meaning that if $Edge(x, y)$ is not true, then $Sub(x, y)$ is not true either. We also show how to insert these bounds in the formulas of T. For example, inserting the upper bound $Edge(x, y)$ for $Sub(x, y)$ and the upper bound $Edge(x, z)$ for $Sub(x, z)$ transforms (5.2) into (5.3).
In the next section, we introduce grounding for FO. In Section 5.2, we present how to add redundant information (bounds) to FO formulas. Next, the results are extended to FO($\cdot$). In Section 5.4 we report on our implementation of a grounding algorithm exploiting bounds. We show by experiments the impact of grounding with bounds compared to grounding without bounds. Finally, we discuss related work.

5.1 Grounding

For the rest of this section, let T be an FO theory over a vocabulary Σ, σ a subvocabulary of Σ and I_σ a finite σ-structure with domain D. We recall the definition of model expansion (MX), i.e., finite model generation with additional data.

Definition 5.1. Let T be an $\mathcal{L}$-theory over a vocabulary Σ, σ a subvocabulary of Σ and I_σ a finite σ-structure. The *model expansion search problem with input* $\langle T, I_\sigma \rangle$ is the problem of computing a Σ-structure M such that $M \models T$ and $M|_\sigma = I_\sigma$.

We denote by $M \models_{I_\sigma} T$ that M is a solution to the model expansion search problem with input $\langle T, I_\sigma \rangle$. I_σ is called the *input structure*, σ the *input vocabulary* and $\Sigma \setminus \sigma$ the *expansion vocabulary*.
Since for every FO theory T, deciding whether T has a model expanding I_σ is in **NP**, this problem can be reduced to a SAT problem T_g in polynomial time.

However, if we want to find models of T expanding I_σ by using a SAT solver, we need a method to translate models of T_g into models of T. Moreover, if we are interested in finding *all* models of T expanding I_σ, a one-to-one correspondence between these models and the models of T_g is needed. In this paper we focus on reductions that preserve all models, which is the setting in the ASP paradigm (Marek and Truszczyński, 1999; Niemelä, 1999).
Let τ be the vocabulary of T_g. To have a one-to-one correspondence between the models of T expanding I_σ and the models of T_g, it should be possible to represent Σ-structures expanding I_σ by τ-structures. The most natural way to accomplish this is by choosing τ such that it contains a symbol $P_{\overline{d}}$ for every $P/n \in \Sigma_P$ and $\overline{d} \in D^n$, and a symbol $F_{\overline{d},d'}$ for every $F/n \in \Sigma_F$ and $(\overline{d}, d') \in D^{n+1}$. A τ-structure making $P_{\overline{d}}$, respectively $F_{\overline{d},d'}$ true then corresponds to a Σ structure M such that $\overline{d} \in P^M$, respectively $F^M(\overline{d}) = d'$. In this manner, every Σ-structure expanding I_σ has a corresponding τ-structure. Vice versa, every τ-structure A such that for every function symbol F/n and $\overline{d} \in D^n$, there is exactly one $d' \in D$ such that $F_{\overline{d},d'}$ is true in A, corresponds to a Σ-structure with the same domains as I_σ. That is, there is a one-to-one correspondence between the τ-structures satisfying for every function symbol F/n and $\overline{d} \in D^n$ the formula

$$\left(\bigvee_{d' \in D} F_{\overline{d},d'}\right) \wedge \left(\bigwedge_{d'_1 \in D} \left(\bigwedge_{d'_2 \in D \setminus d'_1} \neg F_{\overline{d},d'_1} \vee \neg F_{\overline{d},d'_2}\right)\right) \tag{5.4}$$

and the Σ-structures with domain D.
Recall that we denote by $\Sigma^{\mathrm{dom}(I_\sigma)}$ the vocabulary Σ extended with a new constant symbol d, called a *domain constant* for every $d \in D$. For a formula $\varphi[\overline{x}]$ and a tuple $\overline{d}$ of domain constants, we call $\varphi[\overline{x}/\overline{d}]$ an *instance of* φ. For a Σ-structure M expanding I_σ and a formula φ containing domain constants, we denote by $M \models \varphi$ that the expansion of M to $\Sigma^{\mathrm{dom}(I_\sigma)}$ defined by interpreting every domain constant by its corresponding domain element, satisfies φ.

Definition 5.2. Two formulas φ_1 and φ_2 over $\Sigma^{\mathrm{dom}(I_\sigma)}$ are *I_σ-equivalent* if $M\theta \models_{I_\sigma} \varphi_1$ iff $M\theta \models_{I_\sigma} \varphi_2$, for every Σ-structure M and variable assignment θ.

Lemma 5.1. *The following are some results about I_σ-equivalence.*

1. *Two logically equivalent formulas are I_σ-equivalent.*
2. *$\bigwedge_{d \in D} \varphi[x/d]$ is I_σ-equivalent to $\forall x\ \varphi[x]$.*
3. *$\bigvee_{d \in D} \varphi[x/d]$ is I_σ-equivalent to $\exists x\ \varphi[x]$.*
4. *If φ' and ψ' are I_σ-equivalent to respectively φ and ψ, then $(\neg\varphi')$, $(\varphi' \wedge \psi')$, $(\varphi' \vee \psi')$, $(\exists x\ \varphi')$ and $(\forall x\ \varphi')$ are I_σ-equivalent to respectively $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\exists x\ \varphi)$ and $(\forall x\ \varphi)$.*
5. *If ψ is a subformula of φ and is I_σ-equivalent to ψ', then the result of replacing ψ by ψ' in φ is I_σ-equivalent to φ.*

A formula is in *ground normal form* (GNF) if it contains no quantifiers and all its atomic subformulas are of the form $P(d_1, \ldots, d_n)$, $F(d_1, \ldots, d_n) = d$ or $d_1 = d_2$, where $d_1, \ldots, d_n, d$ are domain constants. A theory is in GNF if all its sentences are in GNF. A GNF theory is essentially propositional: by replacing in a GNF theory T every atom $P(\overline{d})$ by $P_{\overline{d}}$, $F(\overline{d}) = d'$ by $F_{\overline{d},d'}$, $d_i = d_j$ by $\top$ or $\bot$ if, respectively, $i = j$ or $i \neq j$, and adding the formula (5.4) for every function symbol F/n and $\overline{d} \in D^n$, we obtain a propositional theory T_{g} such that the models of T and T_{g} correspond. Also note the similarity between GNF and TNF theories.

Definition 5.3. A *grounding for T with respect to I_σ* is a GNF theory T_{g} over $\Sigma^{\mathrm{dom}(I_\sigma)}$ such that T and T_{g} are I_σ-equivalent. T_{g} is called *reduced* if it does not contain symbols of σ.

Grounding algorithms

For the rest of this section, we assume that T is a theory in TNF. As explained in Section 2.1.3, we can make this assumption without loss of generality. Below we introduce, as a reference, the grounding for T with respect to I_σ obtained by the naive grounding algorithm mentioned at the beginning of this chapter. We call this grounding the *full grounding* and define it formally by induction.

Definition 5.4. The *full grounding* $\mathrm{Gr}_{\mathrm{full}}(\varphi, D)$ of a TNF sentence φ with respect to a finite domain D is defined by

$$\mathrm{Gr}_{\mathrm{full}}(\varphi) = \begin{cases} \varphi & \text{if } \varphi \text{ is a literal} \\ \mathrm{Gr}_{\mathrm{full}}(\psi_1) \wedge \mathrm{Gr}_{\mathrm{full}}(\psi_2) & \text{if } \varphi := \psi_1 \wedge \psi_2 \\ \mathrm{Gr}_{\mathrm{full}}(\psi_1) \vee \mathrm{Gr}_{\mathrm{full}}(\psi_2) & \text{if } \varphi := \psi_1 \vee \psi_2 \\ \bigwedge_{d \in D} \mathrm{Gr}_{\mathrm{full}}(\psi[x/d]) & \text{if } \varphi := \forall x\ \psi[x] \\ \bigvee_{d \in D} \mathrm{Gr}_{\mathrm{full}}(\psi[x/d]) & \text{if } \varphi := \exists x\ \psi[x] \end{cases} \tag{5.5}$$

The *full grounding for T with respect to D* is the theory consisting of the full groundings of all sentences in T with respect to D.

We denote the full grounding by $\mathrm{Gr}_{\mathrm{full}}(T, D)$, or by $\mathrm{Gr}_{\mathrm{full}}(T)$ if D is clear from the context. It follows directly from Lemma 5.1 that $\mathrm{Gr}_{\mathrm{full}}(T, D)$ is indeed a grounding for T with respect to I_σ, for every structure I_σ with domain D. The size of the full grounding is exponential in the maximal nesting depth of quantifiers in sentences of T, and polynomial in the size of D.
An inductive definition like (5.5) can be evaluated in a top-down or bottom-up way. Both approaches are applied in current grounders. On the one hand, there are grounders that go top-down through the syntax trees of the sentences in T. When a subformula φ of the form $(\forall x\ \psi[x])$, respectively $(\exists x\ \psi[x])$ is reached, the grounding of $\psi[x/d]$ is constructed for every domain constant d, and then φ is replaced by the conjunction, respectively disjunction, of all these groundings. The grounder of the DLV system (Perri et al., 2007) and the grounders GRINGO (Gebser et al., 2007) and GIDL (Wittocx et al., 2008b) take this approach.

Other grounders go bottom-up through the syntax trees. For each subformula $\varphi[\overline{x}]$ a table is computed consisting of tuples $\overline{d}$ and corresponding groundings of $\varphi[\overline{x}/\overline{d}]$. These tables are computed first for atomic formulas and subsequently for compound formulas. For example, let $\varphi[x, y, z]$ be the formula $\psi[x, y] \wedge \chi[y, z]$ and assume the tables for ψ and χ have been computed. Then the table for φ is computed by taking the natural join of the tables for ψ and χ on the value for y, and constructing the grounding for $\varphi[x/d_x, y/d_y, z/d_z]$ as the (possibly simplified) conjunction of the groundings for $\psi[x/d_x, y/d_y]$ and $\chi[y/d_y, z/d_z]$. Examples of grounders with a bottom-up approach are LPARSE (Syrjänen, 2000; Syrjänen, 2009), KODKOD (Torlak and Jackson, 2007) and MXG (Mitchell et al., 2006).

To obtain a reduced grounding for T with respect to I_σ one could first construct the full grounding and then replace every subformula φ over $\sigma^{\mathrm{dom}(I_\sigma)}$ in it by $\top$ if $I_\sigma \models \varphi$ and by $\bot$ otherwise. The result can further be simplified by recursively replacing $\bot \wedge \psi$ by $\bot$, $\top \wedge \psi$ by ψ, etc. The resulting grounding is the one computed by most current grounding algorithms and is often a lot smaller than the full grounding. We denote it by $\mathrm{Gr}_{\mathrm{red}}(T, I_\sigma)$, or by $\mathrm{Gr}_{\mathrm{red}}(T)$ if I_σ is clear from the context.

Smart grounding algorithms do not use the approach outlined above, but try to avoid creating the full grounding by substituting ground formulas over the input vocabulary σ as soon as possible. For example, a grounder with a top-down approach constructs the grounding of $(\forall x\ \psi[x])$ by grounding all instances $\psi[x/d]$ one by one and then making the conjunction. During this process, all instances $\psi[x/d]$ that are detected to be certainly true are omitted. As soon as an instance $\psi[x/d]$ is detected to be certainly false, $\bot$ is returned as grounding for $(\forall x\ \psi[x])$.

A grounder using the bottom-up approach can reduce the size of the tables it computes by not storing tuples that have some default value, e.g., $\top$, as corresponding grounding. In particular, if $\varphi[\overline{x}]$ is a formula over σ, it only stores the tuples $\overline{d}$ such that $I_\sigma \not\models \varphi[\overline{x}/\overline{d}]$. By reducing the size of the tables in this way, the reduced grounding can be obtained much more efficiently.

5.2 Grounding with bounds

In this section we present our method for reducing grounding size. It is based on computing bounds for subformulas of the input theory T. Each bound for a subformula $\varphi[\overline{x}]$ is a formula over the input vocabulary σ. It describes a set of tuples $\overline{d}$ for which $\varphi[\overline{x}/\overline{d}]$ is certainly true (false) in every model of T expanding any I_σ. The larger the set described by a bound, the more precise the bound is. Observe that the fact that bounds are formulas over σ means that they can be evaluated using the given structure I_σ.

In Section 5.2.1, we formally define bounds. Then we indicate how bounds can be inserted in T to obtain a new theory T'. The reduced grounding of T' is often a lot smaller than the reduced grounding of T. The more precise the inserted bounds are, the smaller the grounding of T' becomes. However, we will

see that T' is in general weaker than T and that additional axioms have to be added to T' to obtain equivalence with T. These additional axioms need to be grounded as well so that, if we are not careful, the total size of the grounded theory does not decrease at all. In Section 5.2.3, we present sufficient conditions on the bounds to guarantee a smaller grounding. In Section 5.2.4, we show how to use the symbolic propagation method to compute bounds that satisfy these conditions.

5.2.1 Bounds

We distinguish between two kinds of bounds.

Definition 5.5. A *certainly true bound (ct-bound) over σ with respect to T for a formula $\varphi[\overline{x}]$* is a formula $\varphi_{\text{ctb}}[\overline{y}]$ over σ such that $\overline{y} \subseteq \overline{x}$ and $T \models \forall \overline{x}\ (\varphi_{\text{ctb}}[\overline{y}] \Rightarrow \varphi[\overline{x}])$. Vice versa, a *certainly false bound (cf-bound) over σ with respect to T for $\varphi[\overline{x}]$* is a formula $\varphi_{\text{cfb}}[\overline{z}]$ over σ such that $\overline{z} \subseteq \overline{x}$ and $T \models \forall \overline{x}\ (\varphi_{\text{cfb}}[\overline{z}] \Rightarrow \neg\varphi[\overline{x}])$.

We do not mention σ and T if they are clear from the context.
Intuitively, a ct-bound φ_{ctb} for $\varphi[\overline{x}]$ provides for every structure I_σ a lower bound for the set of tuples for which φ is true in every model of T expanding I_σ. Indeed, for every $M \models_{I_\sigma} T$ we have that $\{\overline{x} \mid \varphi_{\text{ctb}}\}^{I_\sigma} \subseteq \{\overline{x} \mid \varphi\}^M$. Vice versa, a cf-bound φ_{cfb} provides a lower bound on the set of tuples for which φ is false: $\{\overline{x} \mid \varphi_{\text{cfb}}\}^{I_\sigma} \subseteq \{\overline{x} \mid \neg\varphi\}^M$ for every $M \models_{I_\sigma} T$. Observe that the negation of a ct-bound, respectively cf-bound, gives an upper bound on the set of tuples for which φ is false, respectively true, in at least one model of T expanding I_σ.

Example 5.2 (Example 5.1 ctd.)**.** Denote by φ_1 the subformula $\mathit{Sub}(x,y) \wedge \mathit{Sub}(x,z)$ of T_1. Then $(\neg \mathit{Edge}(x,y) \vee \neg \mathit{Edge}(x,z))$ is a cf-bound over σ_1 with respect to T_1 for φ_1. Indeed, one can derive from (5.1) that T_1 entails

$$\forall x \forall y \forall z\ \ ((\neg \mathit{Edge}(x,y) \vee \neg \mathit{Edge}(x,z)) \Rightarrow \neg\varphi_1)\,.$$

Observe that $\top$ is a ct-bound for every *sentence* of T. Indeed, for every sentence φ of T, $T \models \varphi$ and therefore $T \models \top \Rightarrow \varphi$. Also, $\bot$ is a ct-bound as well as a cf-bound for every *formula*. We call $\bot$ the *trivial bound*. Intuitively, the trivial bound contains no information at all: $\{\overline{x} \mid \bot\}^{I_\sigma} = \emptyset$ for every I_σ and $\overline{x}$. According to the following definition, it is the least precise bound.

Definition 5.6. Let $\psi[\overline{y}]$ and $\chi[\overline{z}]$ be two (ct- or cf-) bounds for $\varphi[\overline{x}]$. We say that $\psi[\overline{y}]$ *is more precise than* $\chi[\overline{z}]$ if $(\forall \overline{x}\ (\chi[\overline{z}] \Rightarrow \psi[\overline{y}]))$ is valid.

If ψ is a more precise bound for $\varphi[\overline{x}]$ than χ, ψ provides a larger lower bound because $\{\overline{x} \mid \chi\}^{I_\sigma} \subseteq \{\overline{x} \mid \psi\}^{I_\sigma}$ for every I_σ.

Definition 5.7. A *c-map* $\mathcal{C}$ for T over σ is a function mapping subformulas φ of T to tuples $(\mathcal{C}^{\text{ctb}}(\varphi), \mathcal{C}^{\text{cfb}}(\varphi))$, where $\mathcal{C}^{\text{ctb}}(\varphi)$ and $\mathcal{C}^{\text{cfb}}(\varphi)$ are respectively a ct- and cf-bound for φ over σ with respect to T.

The notion of precision pointwise extends to c-maps. That is, if $\mathcal{C}_1$ and $\mathcal{C}_2$ are two c-maps for T, then $\mathcal{C}_1$ is more precise than $\mathcal{C}_2$ iff for every subformula φ of T, $\mathcal{C}_1^{\mathrm{ctb}}(\varphi)$ is more precise than $\mathcal{C}_2^{\mathrm{ctb}}(\varphi)$ and $\mathcal{C}_1^{\mathrm{cfb}}(\varphi)$ is more precise than $\mathcal{C}_2^{\mathrm{cfb}}(\varphi)$. Let M be a model of T and $\mathcal{C}$ a c-map for T over σ. From the definition of ct- and cf-bounds it follows immediately that for every subformula $\varphi[\overline{x}]$ of T, both $M \models \forall \overline{x}\ (\mathcal{C}^{\mathrm{ctb}}(\varphi) \Rightarrow \varphi)$ and $M \models \forall \overline{x}\ (\mathcal{C}^{\mathrm{cfb}}(\varphi) \Rightarrow \neg\varphi)$ hold. We say that a structure *satisfies* $\mathcal{C}$ if it has precisely this property.

Definition 5.8. Let $\mathcal{C}$ be a c-map for T over σ. Then the theory $\overline{\mathcal{C}}$ is defined by

$$\begin{aligned}\overline{\mathcal{C}} =&\{\forall \overline{x}\ (\mathcal{C}^{\mathrm{ctb}}(\varphi) \Rightarrow \varphi) \ \mid\ \varphi[\overline{x}] \text{ is a subformula of } T\}\\ &\cup \{\forall \overline{x}\ (\mathcal{C}^{\mathrm{cfb}}(\varphi) \Rightarrow \neg\varphi) \ \mid\ \varphi[\overline{x}] \text{ is a subformula of } T\}.\end{aligned}$$

A structure I *satisfies* $\mathcal{C}$ if $I \models \overline{\mathcal{C}}$.

Clearly, if $\mathcal{C}$ is a c-map for T over σ and $M \models_{I_\sigma} T$, then $M \models \overline{\mathcal{C}}$. We call two formulas $\varphi[\overline{x}]$ and $\psi[\overline{x}]$ $\mathcal{C}$*-equivalent* if $\{\overline{x} \mid \varphi\}^I = \{\overline{x} \mid \psi\}^I$ for each structure I that satisfies $\mathcal{C}$. Equivalently, φ and ψ are $\mathcal{C}$-equivalent if $\overline{\mathcal{C}} \models \forall \overline{x}\ (\varphi \Leftrightarrow \psi)$.
A c-map is *inconsistent* if some formula φ is both certainly true and false for some tuple, according to that c-map:

Definition 5.9. A c-map $\mathcal{C}$ for T over σ is *inconsistent* if $\exists \overline{x}\ (\mathcal{C}^{\mathrm{ctb}}(\varphi) \wedge \mathcal{C}^{\mathrm{cfb}}(\varphi))$ is valid for some subformula $\varphi[\overline{x}]$ of T. A c-map $\mathcal{C}$ is I_σ*-inconsistent* if $I_\sigma \models \exists \overline{x}\ (\mathcal{C}^{\mathrm{ctb}}(\varphi) \wedge \mathcal{C}^{\mathrm{cfb}}(\varphi))$ for some subformula of T.

Proposition 5.2. *If there exists an I_σ-inconsistent c-map for T over σ, then $M \not\models_{I_\sigma} T$ for every structure M. If there exists an inconsistent c-map for T over σ, then $M \not\models_{I_\sigma} T$ for every structure M and I_σ.*

Proof. Let $\mathcal{C}$ be an I_σ-inconsistent c-map for T over σ and $\varphi[\overline{x}]$ a subformula of T such that $I_\sigma \models \exists \overline{x}\ (\mathcal{C}^{\mathrm{ctb}}(\varphi) \wedge \mathcal{C}^{\mathrm{cfb}}(\varphi))$. Then there exists a tuple of domain elements $\overline{d}$ such that $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}^{\mathrm{ctb}}(\varphi)$ and $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}^{\mathrm{cfb}}(\varphi)$. Assume towards a contradiction that $M \models_{I_\sigma} T$. Then $M \models \overline{\mathcal{C}}$, and hence $M[\overline{x}/\overline{d}] \models \mathcal{C}^{\mathrm{ctb}}(\varphi) \Rightarrow \varphi$ and $M[\overline{x}/\overline{d}] \models \mathcal{C}^{\mathrm{cfb}}(\varphi) \Rightarrow \neg\varphi$. Since $M|_\sigma = I_\sigma$, it follows that $M[\overline{x}/\overline{d}] \models \varphi$ and $M[\overline{x}/\overline{d}] \models \neg\varphi$. This is a contradiction.
To prove the second statement, let $\mathcal{C}$ be an inconsistent c-map for T over σ. Then $\mathcal{C}$ is a also an I_σ-inconsistent c-map for every σ-structure I_σ. As such, for any I_σ there is no model of T expanding I_σ. ■

5.2.2 C-transformation

For the rest of this section, fix a c-map $\mathcal{C}$ for T over σ. We now show how to insert the bounds of $\mathcal{C}$ into the sentences of T. The insertion is based on the following lemma.

Lemma 5.3. *Let $\varphi[\overline{x}]$ be a subformula of T. Then φ is $\mathcal{C}$-equivalent to $(\varphi \vee \mathcal{C}^{\mathrm{ctb}}(\varphi))$ and to $(\varphi \wedge \neg\mathcal{C}^{\mathrm{cfb}}(\varphi))$.*

Proof. We have to prove that $\overline{\mathcal{C}} \models \forall \overline{x}\ (\varphi \Leftrightarrow (\varphi \vee \mathcal{C}^{\mathrm{ctb}}(\varphi)))$ and $\overline{\mathcal{C}} \models \forall \overline{x}\ (\varphi \Leftrightarrow (\varphi \wedge \neg\mathcal{C}^{\mathrm{cfb}}(\varphi)))$. The former immediately follows from the fact that $\overline{\mathcal{C}} \models \forall \overline{x}\ (\mathcal{C}^{\mathrm{ctb}}(\varphi) \Rightarrow \varphi)$, the latter from the fact that $\overline{\mathcal{C}} \models \forall \overline{x}\ (\mathcal{C}^{\mathrm{cfb}}(\varphi) \Rightarrow \neg\varphi)$. ■

As a corollary of lemma 5.3 we have the following lemma.

Lemma 5.4. *Let ψ be a sentence of T and φ a subformula of ψ. If ψ' is the result of replacing the subformula φ in ψ by $(\varphi \vee \mathcal{C}^{\mathrm{ctb}}(\varphi))$, by $(\varphi \wedge \neg\mathcal{C}^{\mathrm{cfb}}(\varphi))$ or by $((\varphi \wedge \neg\mathcal{C}^{\mathrm{cfb}}(\varphi)) \vee \mathcal{C}^{\mathrm{ctb}}(\varphi))$, then $M \models \psi$ iff $M \models \psi'$ for every M that satisfies $\mathcal{C}$.*

Observe that if $\mathcal{C}^{\mathrm{ctb}}(\varphi) = \mathcal{C}^{\mathrm{cfb}}(\varphi) = \bot$, then both $(\varphi \vee \mathcal{C}^{\mathrm{ctb}}(\varphi))$ and $(\varphi \wedge \neg\mathcal{C}^{\mathrm{cfb}}(\varphi))$ are logically equivalent to φ. Hence, in this case the sentence ψ' in Lemma 5.4 is essentially the sentence ψ. In other words, adding trivial bounds to a sentence ψ does not change the sentence at all.
The bounds assigned by $\mathcal{C}$ can be "inserted" in T by applying the transformation of Lemma 5.4 to all subformulas of T. The result is called a c-transformation of T, and is formally defined as follows.

Definition 5.10. A *c-transformation* of a subformula φ of T with respect to $\mathcal{C}$, denoted $\mathcal{C}\langle\varphi\rangle$, is the formula $(\varphi' \wedge \neg\mathcal{C}^{\mathrm{cfb}}(\varphi)) \vee \mathcal{C}^{\mathrm{ctb}}(\varphi)$ where φ' is defined by

$$\varphi' := \begin{cases} \varphi & \text{if } \varphi \text{ is an atom} \\ \neg\mathcal{C}\langle\psi\rangle & \text{if } \varphi \text{ is equal to } \neg\psi \\ \mathcal{C}\langle\psi\rangle \wedge \mathcal{C}\langle\chi\rangle & \text{if } \varphi \text{ is equal to } \psi \wedge \chi \\ \mathcal{C}\langle\psi\rangle \vee \mathcal{C}\langle\chi\rangle & \text{if } \varphi \text{ is equal to } \psi \vee \chi \\ \exists x\ \mathcal{C}\langle\psi\rangle & \text{if } \varphi \text{ is equal to } \exists x\ \psi \\ \forall x\ \mathcal{C}\langle\psi\rangle & \text{if } \varphi \text{ is equal to } \forall x\ \psi \end{cases}$$

A *c-transformation $\mathcal{C}\langle T\rangle$ of T with respect to $\mathcal{C}$* consists of a c-transformation with respect to $\mathcal{C}$ of every sentence of T.

From Lemma 5.4, we derive the following.

Lemma 5.5. *T and $\mathcal{C}\langle T\rangle$ are $\mathcal{C}$-equivalent.*

In general T and $\mathcal{C}\langle T\rangle$ are not logically equivalent. $\mathcal{C}\langle T\rangle$ may have models that do not satisfy $\mathcal{C}$, and therefore cannot be models of T. For example, let $\mathcal{C}$ be the c-map that assigns $(\top, \bot)$ to every sentence and $(\bot, \bot)$ to every other subformula of T. Then all sentences in $\mathcal{C}\langle T\rangle$ are of the form $\varphi \vee \top$ and hence $\mathcal{C}\langle T\rangle$ simplifies to $\top$, which is in general not equivalent to T. To obtain from $\mathcal{C}\langle T\rangle$ a theory that is equivalent to T, we must add $\overline{\mathcal{C}}$.

Theorem 5.6. *If $\mathcal{C}$ is a c-map for T over σ and $\overline{\mathcal{C}}$ the theory defined in Definition 5.8, then $\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}$ is logically equivalent to T.*

Proof. Let M be a model of T. Then $M \models \overline{\mathcal{C}}$, and because of Lemma 5.4, $M \models \mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}$. On the other hand, if $M \models \mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}$, then by Lemma 5.4, $M \models T$. ■

Corollary 5.7. *If $\mathcal{C}$ is a c-map for T over σ, then T and $\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}$ are I_σ-equivalent for any σ-structure I_σ.*

5.2.3 Atom-based and atom-equal c-maps

Corollary 5.7 states that we can compute a grounding for T with respect to I_σ by first computing a c-map $\mathcal{C}$ for T over σ and then grounding $\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}$. This approach is beneficial if the reduced grounding of $\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}$ is smaller than the reduced grounding of T, and can be constructed at least as fast. In general these conditions are not satisfied. The more precise c-map $\mathcal{C}$ is, the smaller the reduced grounding of $\mathcal{C}\langle T\rangle$ becomes, but the larger the reduced grounding of $\overline{\mathcal{C}}$ is.

Proposition 5.8. *If $\mathcal{C}_1$ is more precise than $\mathcal{C}_2$, then $Gr_{red}(\mathcal{C}_1\langle T\rangle)$ is smaller than $Gr_{red}(\mathcal{C}_2\langle T\rangle)$. Moreover, every subformula that occurs in $Gr_{red}(\mathcal{C}_1\langle T\rangle)$ also occurs in $Gr_{red}(\mathcal{C}_2\langle T\rangle)$.*

Proof. (Sketch) Let $\varphi[\overline{x}]$ be a subformula of T and $\overline{d}$ a tuple of domain elements. It suffices to show that if $\mathcal{C}_2\langle\varphi\rangle[\overline{x}/\overline{d}]$ is replaced by $\top$, respectively $\bot$ when grounding, then this is also the case for $\mathcal{C}_1\langle\varphi\rangle[\overline{x}/\overline{d}]$. This can be proven by induction. For the base case, assume φ is an atom. Then $\mathcal{C}_2\langle\varphi\rangle[\overline{x}/\overline{d}]$ is the formula $(\varphi \wedge \neg\mathcal{C}_2^{\text{cfb}}(\varphi)) \vee \mathcal{C}_2^{\text{ctb}}(\varphi)$. If this formula is replaced by $\top$ or $\bot$ when grounding, there are three possibilities: φ is a formula over σ, $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}_2^{\text{ctb}}(\varphi)$ or $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}_2^{\text{cfb}}(\varphi)$. Since $\mathcal{C}_1$ is more precise than $\mathcal{C}_2$, $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}_2^{\text{ctb}}(\varphi)$ implies $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}_1^{\text{ctb}}(\varphi)$ and $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}_2^{\text{cfb}}(\varphi)$ implies $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}_1^{\text{cfb}}(\varphi)$. We conclude that if $\mathcal{C}_2\langle\varphi\rangle[\overline{x}/\overline{d}]$ is replaced by $\top$ or $\bot$ when grounding, then this is also the case for $\mathcal{C}_1\langle\varphi\rangle[\overline{x}/\overline{d}]$. The inductive case is similar. ■

Proposition 5.9. *If $\mathcal{C}_1$ is more precise than $\mathcal{C}_2$, then $Gr_{red}(\overline{\mathcal{C}_1})$ is larger than $Gr_{red}(\overline{\mathcal{C}_2})$.*

Proof. (Sketch) Every sentence in $\overline{\mathcal{C}_1}$ is of the form $\forall\overline{x}\ (\mathcal{C}_1^{\text{ctb}}(\varphi) \Rightarrow \varphi)$ or $\forall\overline{x}\ (\mathcal{C}_1^{\text{cfb}}(\varphi) \Rightarrow \neg\varphi)$. The number of instances of $\mathcal{C}_1^{\text{ctb}}(\varphi) \Rightarrow \varphi$ in the reduced grounding of $\overline{\mathcal{C}_1}$ is equal to the number of $\overline{d}$ such that $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}_1^{\text{ctb}}(\varphi)$. Similarly for $\mathcal{C}_1^{\text{cfb}}(\varphi) \Rightarrow \neg\varphi$. Since $\mathcal{C}_2$ is less precise than $\mathcal{C}_1$, the number of instances in $\text{Gr}_{\text{red}}(\overline{\mathcal{C}_2})$ of the corresponding sentences $\forall\overline{x}\ (\mathcal{C}_2^{\text{ctb}}(\varphi) \Rightarrow \varphi)$ and $\forall\overline{x}\ (\mathcal{C}_2^{\text{cfb}}(\varphi) \Rightarrow \neg\varphi)$ is smaller. ■

A c-map that is useful to reduce grounding size should not be too precise, in order to avoid a large theory $\text{Gr}_{\text{red}}(\overline{\mathcal{C}})$, but still be precise enough to decrease the size of $\text{Gr}_{\text{red}}(\mathcal{C}\langle T\rangle)$. In this section, we present sufficient conditions to ensure these properties. We first define a class of c-maps that "avoid" a blow-up of $\text{Gr}_{\text{red}}(\overline{\mathcal{C}})$ by ensuring $\overline{\mathcal{C}}$ can be replaced by an equivalent, smaller and easy-to-find theory $\overline{\mathcal{C}}_A$. As such, $\text{Gr}_{\text{red}}(\overline{\mathcal{C}})$ can be replaced by the smaller theory $\text{Gr}_{\text{red}}(\overline{\mathcal{C}}_A)$. In the class we present, $\overline{\mathcal{C}}_A$ is a subset of $\overline{\mathcal{C}}$, namely the set of sentences in $\overline{\mathcal{C}}$ that stem from the atomic subformulas of T.

Definition 5.11. Define the theory $\overline{\mathcal{C}}_A$ by

$$\begin{aligned}\overline{\mathcal{C}}_A =&\{\forall \overline{x}\ (\mathcal{C}^{\mathrm{ctb}}(\varphi) \Rightarrow \varphi)\ \mid\ \varphi[\overline{x}] \text{ is an atomic subformula of } T\}\\ &\cup \{\forall \overline{x}\ (\mathcal{C}^{\mathrm{cfb}}(\varphi) \Rightarrow \neg\varphi)\ \mid\ \varphi[\overline{x}] \text{ is an atomic subformula of } T\}.\end{aligned}$$

We call $\mathcal{C}$ *atom-based* if $\overline{\mathcal{C}}_A \models \overline{\mathcal{C}}$.

Example 5.3 (Example 5.1 ctd.)**.** Let $\mathcal{C}_2$ be the c-map that assigns the pair $(\bot, \neg(\mathit{Edge}(x,y) \wedge \mathit{Edge}(x,z)))$ to $(\mathit{Sub}(x,y) \wedge \mathit{Sub}(x,z))$ and the pair $(\bot,\bot)$ to every other subformula. $\mathcal{C}_2$ is not atom-based, since $(\overline{\mathcal{C}}_2)_A$ is equivalent to $\top$, while $\overline{\mathcal{C}}_2$ contains the sentence

$$\forall x \forall y \forall z\ (\neg(\mathit{Edge}(x,y) \wedge \mathit{Edge}(x,z)) \Rightarrow \neg(\mathit{Sub}(x,y) \wedge \mathit{Sub}(x,z))). \tag{5.6}$$

Let $\mathcal{C}_3$ be the c-map that assigns $(\bot, \neg\mathit{Edge}(x,y))$ to $\mathit{Sub}(x,y)$, $(\bot, \neg\mathit{Edge}(x,z))$ to $\mathit{Sub}(x,z)$ and corresponds to $\mathcal{C}_2$ on all other subformulas of T_1. $\mathcal{C}_3$ is atom-based. Indeed, $(\overline{\mathcal{C}}_3)_A$ consists of the sentences

$$\forall x \forall y\ (\neg\mathit{Edge}(x,y) \Rightarrow \neg\mathit{Sub}(x,y)) \tag{5.7}$$

$$\forall x \forall z\ (\neg\mathit{Edge}(x,z) \Rightarrow \neg\mathit{Sub}(x,z)) \tag{5.8}$$

and $\overline{\mathcal{C}}_3$ consists of the sentences (5.6), (5.7) and (5.8). The conjunction of (5.7) and (5.8) implies (5.6), and therefore, $(\overline{\mathcal{C}}_3)_A \models \overline{\mathcal{C}}_3$.

Clearly, a c-map assigning $(\bot,\bot)$ to every non-atomic subformula of T is an example of an atom-based c-map. As such, any c-map can be transformed into an atom-based one by replacing every bound assigned to a non-atomic subformula by $\bot$. In the next section, we show how to compute more interesting atom-based c-maps.
Observe that $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$ contains only unit clauses. Combining the definition of atom-based c-map and Theorem 5.6 immediately gives the following result.

Proposition 5.10. *Let $\mathcal{C}$ be an atom-based c-map for T over σ. Then T and $\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}_A$ are equivalent, and hence I_σ-equivalent for every σ-structure I_σ.*

To obtain small groundings using bounds, it is important that the information in the bounds is exploited wherever possible. In particular, if a ct- or cf-bound ψ is assigned to an atom $P(\overline{x})$, then a similar bound should be assigned to every other atom of the form $P(\overline{y})$. We call a c-map *atom-equal* if it has exactly this property for all atomic subformulas of T. That is, $\mathcal{C}$ is atom-equal if it assigns essentially the same bounds to atomic subformulas over the same predicate or function symbol.

Definition 5.12. A c-map $\mathcal{C}$ for a TNF theory T over σ is *atom-equal* if for every predicate symbol P/n there exist formulas $\varphi_P^{\mathrm{ctb}}[x_1,\ldots,x_n]$ and $\varphi_P^{\mathrm{cfb}}[x_1,\ldots,x_n]$ such that for every atom $P(y_1,\ldots,y_n)$ that occurs in T, $\mathcal{C}^{\mathrm{ctb}}(P(y_1,\ldots,y_n)) = \varphi_P^{\mathrm{ctb}}[x_1/y_1,\ldots,x_n/y_n]$ and $\mathcal{C}^{\mathrm{cfb}}(P(y_1,\ldots,y_n)) = \varphi_P^{\mathrm{cfb}}[x_1/y_1,\ldots,x_n/y_n]$, and similarly for function symbols.

Example 5.4 (Example 5.1 ctd.)**.** Let T_2 be the theory obtained by adding the sentence $(\exists w\ \mathit{Sub}(w,w))$ to T_1. Let $\mathcal{C}_4$ be the c-map for T_2 that assigns $(\bot, \neg \mathit{Edge}(u,v))$ to $\mathit{Sub}(u,v)$, $(\bot, \neg \mathit{Edge}(x,y))$ to $\mathit{Sub}(x,y)$, $(\bot, \neg \mathit{Edge}(x,z))$ to $\mathit{Sub}(x,z)$, $(\bot, \neg \mathit{Edge}(w,w))$ to $\mathit{Sub}(w,w)$ and $(\bot,\bot)$ to all other subformulas of T_2. Then $\mathcal{C}_4$ is atom-equal. Indeed, the only predicate that occurs more than once in T_2 is Sub, and to all occurrences of Sub, $\mathcal{C}_4$ assigns essentially the same bounds. Indeed, we can take $\varphi^{\mathrm{ctb}}_{\mathit{Sub}} = \bot$ and $\varphi^{\mathrm{cfb}}_{\mathit{Sub}} = \neg \mathit{Edge}(x_1,x_2)$, where $\varphi^{\mathrm{ctb}}_{\mathit{Sub}}$ and $\varphi^{\mathrm{cfb}}_{\mathit{Sub}}$ are the formulas mentioned in definition 5.12.

For an atom-equal c-map $\mathcal{C}$, $\overline{\mathcal{C}}_A$ in general contains many equivalent sentences. For example, for the c-map $\mathcal{C}_4$ as in Example 5.4, $(\overline{\mathcal{C}}_4)_A$ contains amongst others, the equivalent sentences (5.7) and (5.8). It also contains $\forall w\ (\neg \mathit{Edge}(w,w) \Rightarrow \neg \mathit{Sub}(w,w))$, which is implied by (5.7). As a result, if $\mathcal{C}$ is an atom-equal c-map, grounding $\overline{\mathcal{C}}_A$ in a naive way yields a grounding that contains several formulas more than once. In the following proposition, we assume this redundancy is removed. In other words, we assume a grounding algorithm for $\overline{\mathcal{C}}_A$ that never adds the same GNF formula more than once to the grounding. This can be accomplished by grounding instead of $\overline{\mathcal{C}}_A$ the sentences $\forall \overline{x}\ (\varphi^{\mathrm{ctb}}_P \Rightarrow P(\overline{x}))$ and $\forall \overline{x}\ (\varphi^{\mathrm{cfb}}_P \Rightarrow \neg P(\overline{x}))$ for every predicate symbol P, where φ^{ctb}_P and φ^{cfb}_P are as in Definition 5.12.

Proposition 5.11. *Let $\mathcal{C}$ be an atom-based, atom-equal c-map for a TNF theory T. If T has a model expanding I_σ, then $\mathit{Gr}_{\mathit{red}}(\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}_A)$ is smaller than $\mathit{Gr}_{\mathit{red}}(T)$.*

In the proof, we denote the size of a theory T_{g} by $|T_{\mathrm{g}}|$.

Proof. The outline of this proof is as follows. First, we show that every subformula that occurs in $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle)$, occurs in $\mathrm{Gr}_{\mathrm{red}}(T)$. Then, we prove that no atom occurring in $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$ occurs in $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle)$. Next, we show that every atom occurring in $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$ occurs at least once in $\mathrm{Gr}_{\mathrm{red}}(T)$. Since we assumed $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$ does not contain any formula more than once, it follows that $|\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle)| \leq |\mathrm{Gr}_{\mathrm{red}}(T)| - |\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)|$, which concludes the proof.
We apply Proposition 5.8 to show that every subformula of $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle)$ occurs in $\mathrm{Gr}_{\mathrm{red}}(T)$: if $\mathcal{C}'$ is the trivial c-map, then $\mathrm{Gr}_{\mathrm{red}}(T)$ is equal to $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}'\langle T\rangle)$, and clearly $\mathcal{C}$ is more precise than $\mathcal{C}'$.
We now show that none of the atoms occurring in $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$ can occur in $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle)$. Let $P(\overline{d})$ be an atom occurring in $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle)$. Then there is an atomic subformula $P(\overline{x})$ of T such that $\overline{d} \notin \{\overline{x} \mid \mathcal{C}^{\mathrm{ctb}}(P(\overline{x}))\}^{I_\sigma}$ and $\overline{d} \notin \{\overline{x} \mid \mathcal{C}^{\mathrm{cfb}}(P(\overline{x}))\}^{I_\sigma}$. Because $\mathcal{C}$ is atom-equal, it follows that for any subformula $P(\overline{y})$ occurring in T, neither $\overline{d} \in \{\overline{y} \mid \mathcal{C}^{\mathrm{ctb}}(P(\overline{y}))\}^{I_\sigma}$ nor $\overline{d} \in \{\overline{y} \mid \mathcal{C}^{\mathrm{cfb}}(P(\overline{y}))\}^{I_\sigma}$. Therefore $P(\overline{d})$ does not occur in $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$.
It remains to show that every atom that occurs in $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$ also occurs in $\mathrm{Gr}_{\mathrm{red}}(T)$. Let M be a model of $\mathrm{Gr}_{\mathrm{red}}(T)$. Such a model exists because we assumed that T has a model expanding I_σ. Let $P(\overline{d})$ be an atom that does not occur in $\mathrm{Gr}_{\mathrm{red}}(T)$. If P is a predicate of the input vocabulary, then $P(\overline{d})$ does not occur in $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$ either. If on the other hand, P is in the expansion vocabulary, then both $M[P(\overline{d})/\mathbf{t}]$ and $M[P(\overline{d})/\mathbf{f}]$ are models of $\mathrm{Gr}_{\mathrm{red}}(T)$.

Since $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}_A)$ is I_σ-equivalent to $\mathrm{Gr}_{\mathrm{red}}(T)$ and $P \notin \sigma$, it follows that $M[P(\overline{d})/\mathbf{t}] \models \mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$ and $M[P(\overline{d})/\mathbf{f}] \models \mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$. Because $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$ only contains unit clauses, we conclude that $P(\overline{d})$ does not occur in $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}}_A)$. ■

We now have the following algorithm to create a small grounding for T with respect to I_σ: First compute an atom-based, atom-equal c-map $\mathcal{C}$ for T over σ (We will present an algorithm for this in Section 5.2.4). If $\mathcal{C}$ is I_σ-inconsistent, output $\bot$ and stop. Else, output $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}_A)$.
It follows from Propositions 5.2 and 5.10 that the result of this algorithm is indeed a grounding for T with respect to I_σ. Observe that the first step of this algorithm is independent of I_σ. If one has to solve several model expansion problems with a fixed input theory T and input vocabulary σ, but varying I_σ, it suffices to compute $\mathcal{C}$ only once.
To perform the last step of the algorithm, one could apply any off-the-shelf grounder on input $\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}_A$.

5.2.4 Computing bounds

An atom-based, atom-equal c-map $\mathcal{C}$ for an FO theory T can be obtained by symbolic propagation. This is done by starting from the symbolic Σ-structure $\tilde{\Phi}_\sigma$ over the input vocabulary σ, that assigns

- $(\{\overline{x} \mid P(\overline{x})\}, \{\overline{x} \mid \neg P(\overline{x})\})$ to every predicate symbol P in σ;
- $(\{(\overline{x}, y) \mid F(\overline{x}) = y\}, \{(\overline{x}, y) \mid F(\overline{x}) \neq y\})$ to every function symbol F in σ;
- $(\{\overline{x} \mid \bot\}, \{\overline{x} \mid \bot\})$ to all predicate and function symbols in the expansion vocabulary $\Sigma \setminus \sigma$.

Next, symbolic constraint propagation is applied on input T and $\tilde{\Phi}_\sigma$ to obtain a more precise symbolic structure $\tilde{\Psi}$. $\tilde{\Psi}$ can be seen as a function that maps each subformula φ of T to the pair of σ-formulas $\tilde{\Psi}(\varphi)$, i.e., to the pair $(\tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{ct}}), \tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{cf}}))$. We show that this function is an atom-based, atom-equal c-map for T.

Proposition 5.12. *Let $\tilde{\Psi}$ be the result of applying symbolic propagation on T and $\tilde{\Phi}_\sigma$. Then $\tilde{\Psi}$ is a c-map for T over σ. Moreover, it is atom-based and atom-equal.*

Proof. To show that $\tilde{\Psi}$ is a c-map, we need to prove that $T \models \forall \overline{x}\ (\tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{ct}}) \Rightarrow \varphi)$ and $T \models \forall \overline{x}\ (\tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{cf}}) \Rightarrow \neg\varphi)$ for every subformula $\varphi[\overline{x}]$ of T. Let M a model of T and let $\tilde{\Phi}_\sigma$ be the structure defined above. Then $M(\tilde{\Phi}_\sigma)|_\sigma = M|_\sigma$ and $M(\tilde{\Phi}_\sigma)|_{\Sigma\setminus\sigma} = \bot^{\leq_p}$. Therefore $M(\tilde{\Phi}_\sigma) \leq_p M$. Since $\tilde{\Psi}$ is obtained from $\tilde{\Phi}_\sigma$ by constraint propagation on T, it follows from Definition 4.8 that also $M(\tilde{\Psi}) \leq_p M$. Hence, by Lemma 4.15, $(M[\overline{x}/\overline{d}](\tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{ct}})), M[\overline{x}/\overline{d}](\tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{cf}}))) \leq_p M[\overline{x}/\overline{d}](\varphi)$ for every tuple $\overline{d}$ of domain elements. It follows that $M[\overline{x}/\overline{d}] \models \tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{ct}}) \Rightarrow \varphi$ and $M[\overline{x}/\overline{d}] \models \tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{cf}}) \Rightarrow \neg\varphi$ for every $\overline{d}$ and model M of T. We conclude that $\tilde{\Psi}$ is a c-map for T.

We now show that $\tilde{\Psi}$ is atom-equal. Let P be a predicate and denote by $(\{(x_1,\ldots,x_n) \mid \varphi_1\},\{(x_1,\ldots,x_n) \mid \varphi_2\})$ the pair of queries assigned by $\tilde{\Psi}$ to P. Then for every atomic subformula $P(y_1,\ldots,y_n)$ of T, $\tilde{\Psi}(P(y_1,\ldots,y_n))$ is the pair $(\varphi_1[x_1/y_1,\ldots,x_n/y_n],\varphi_2[x_1/y_1,\ldots,x_n/y_n])$. Hence, φ_1 and φ_2 can serve as, respectively, the formulas φ_P^{ctb} and φ_P^{cfb} from Definition 5.12. The same argument applies for functions. It follows that $\tilde{\Psi}$ is atom-equal.
To show that $\tilde{\Psi}$ is atom-based, we have to prove that for any structure I, if I satisfies the sentences $\forall\overline{x}\ (\tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{ct}}) \Rightarrow \varphi)$ and $\forall\overline{x}\ (\tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{cf}}) \Rightarrow \neg\varphi)$ for every atomic subformula φ of T, I also satisfies these sentences for every non-atomic formula φ of T. The proof is by a simple structural induction. We prove one case, all other cases are similar. Let I be a structure and $\varphi[\overline{x}]$ the sentence $(\forall y\ \psi[\overline{x},y])$. Assume that $I \models \forall\overline{x}\forall y\ (\tilde{\Psi}^{\mathrm{tf}}(\psi^{\mathrm{ct}}) \Rightarrow \psi)$. It follows that $I \models \forall\overline{x}\ ((\forall y\ \tilde{\Psi}^{\mathrm{tf}}(\psi^{\mathrm{ct}})) \Rightarrow (\forall y\ \psi))$, and hence $I \models \forall\overline{x}\ (\tilde{\Psi}^{\mathrm{tf}}(\varphi^{\mathrm{ct}}) \Rightarrow \varphi)$. ■

We illustrate the above method to compute a c-map on our running example.

Example 5.5 (Example 5.1 ctd.)**.** The method starts from symbolic structure $\tilde{\Phi}_\sigma$ assigning $(\{(x,y) \mid \mathit{Edge}(x,y)\},\{(x,y) \mid \neg\mathit{Edge}(x,y)\})$ to *Edge* and $(\{\overline{x} \mid \bot\},\{\overline{x} \mid \bot\})$ to *Sub*. Applying the symbolic propagation algorithm (Algorithm 4.3) on T_1 and $\tilde{\Phi}_\sigma$ leads to symbolic structure $\tilde{\Psi}$ assigning $(\{(x,y) \mid \bot\},\{(x,y) \mid \neg\mathit{Edge}(x,y)\})$ to *Sub*. Viewed as a c-map, $\tilde{\Psi}$ assigns $(\neg\mathit{Edge}(x,y)\vee \neg\mathit{Edge}(x,z))$ as cf-bound for the subformula $(\mathit{Sub}(x,y)\wedge\mathit{Sub}(x,z))$ of T_1. As a result, the c-transformation of this subformula for c-map $\tilde{\Psi}$ is given by

$$((\mathit{Sub}(x,y)\wedge\mathit{Edge}(x,y))\wedge(\mathit{Sub}(x,z)\wedge\mathit{Edge}(x,z)))\wedge(\mathit{Edge}(x,y)\wedge\mathit{Edge}(x,z)).$$

Note that this formula contains repeated constraints $\mathit{Edge}(x,y)$ and $\mathit{Edge}(x,z)$ on the variables x,y and z. In general the c-maps corresponding to symbolic structures produce many such repetitions. These could easily be eliminated to speed up the grounding process, but it depends on the used grounding algorithm which ones are best deleted. See Section 5.4.1 for an example.

5.3 Grounding FO(·) with bounds

In this section, we extend our results about grounding FO with bounds to FO(·). We mainly focus on FO(ID), since handling the other extensions is straightforward. To facilitate the presentation, we assume that no function symbols occur in the head of a definition rule, no predicate of the input vocabulary is defined by a definition, no expansion predicate is defined by more than one definition and every rule body is in TNF. The results of Section 3.5.3 allow to make this assumption without loss of generality.

5.3.1 Grounding inductive definitions with bounds

As we showed in Section 5.1, every FO model expansion problem can be reduced by grounding to an equivalent SAT problem. Similarly, every FO(ID) model

expansion problem can be reduced to an equivalent satisfiability problem in the propositional fragment of FO(ID), called a SAT[ID] problem. The problem can then be solved by a SAT[ID] solver. Currently there exist three such solvers. IDSAT (Pelov and Ternovska, 2005) works by translating a SAT[ID] problem into an equivalent SAT problem and then calls a SAT solver. MIDL (Mariën et al., 2007b) and MINISAT(ID) (Mariën et al., 2008) take a native approach. Mariën (2009) provides details on the specific form of propositional FO(ID) theories accepted by these solvers, and a method to transform arbitrary propositional FO(ID) theories into this form.
We formally define grounding for FO(ID). Let T be an FO(ID) theory. As for FO, a grounding T_g for T with respect to I_σ is a propositional FO(ID) theory that is I_σ-equivalent to T. We extend the notion of full and reduced grounding to definitions.

Definition 5.13. The *full grounding of a rule* $\forall \overline{x}\ P(\overline{x}) \leftarrow \varphi$ with respect to finite domain D is the set $\{P(\overline{d}) \leftarrow \mathrm{Gr}_{\mathrm{full}}(\varphi[\overline{x}/\overline{d}]) \mid \overline{d} \in D^{|\overline{x}|}\}$. Similarly, the *reduced grounding of* $\forall \overline{x}\ (P(\overline{x}) \leftarrow \varphi)$ with respect to I_σ is the set $\{P(\overline{d}) \leftarrow \mathrm{Gr}_{\mathrm{red}}(\varphi[\overline{x}/\overline{d}], I_\sigma) \mid \overline{d} \in D^{|\overline{x}|}\}$. The *full (reduced) grounding of a definition* Δ is the union of the full (reduced) groundings of all rules in Δ. The full (reduced) grounding of an FO(ID) theory T is the set of the full (reduced) groundings of all sentences and definitions in T.

Bounds for definitions

We now extend results about grounding FO with bounds to grounding FO(ID) with bounds. The major difficulty is that a straightforward extension of the concept of c-transformation to definitions is not equivalence preserving. We investigate sufficient conditions on c-maps to ensure that equivalence is preserved.

Definition 5.14. A formula φ is a *subformula* of an FO(ID) theory T if it is a subformula of a sentence in T or a subformula of a rule body in a definition of T. A *c-map* for T over σ is an assignment of a ct- and cf-bound over σ to every subformula of T.

Note that a c-map does not assign bounds to heads of rules in a definition. The notions of atom-based c-map and atom-equal c-map extend to c-maps for FO(ID). C-maps for FO(ID) theories can be computed like c-maps for FO theories: it suffices to apply symbolic propagation on the completion of the input theory T and a suitable symbolic structure $\tilde{\Phi}_\sigma$ to obtain a more precise symbolic structure $\tilde{\Psi}$, which can be viewed as a mapping from subformulas to pairs of bounds. Proposition 5.12 extends to FO(ID): the computed mapping $\tilde{\Psi}$ is an atom-based, atom-equal c-map for T.
In order to use a c-map for grounding, the definition of c-transformation is lifted to FO(ID) theories.

Definition 5.15. Let $\mathcal{C}$ be a c-map for a theory T and Δ a definition in T. The *c-transformation of a rule* $\forall \overline{x}\ (P(\bar{t}) \leftarrow \varphi)$ *of* Δ is given by $\forall \overline{x}\ (P(\bar{t}) \leftarrow \mathcal{C}\langle\varphi\rangle)$. The c-transformation $\mathcal{C}\langle\Delta\rangle$ of a definition Δ is the set of c-transformations of

rules in Δ. The c-transformation of T is the set of the c-transformations of the formulas and definitions in T.

We also lift the notion of $\mathcal{C}$-equivalence to definitions.

Definition 5.16. Two definitions Δ_1 and Δ_2 are *$\mathcal{C}$-equivalent* if for every structure I that satisfies $\overline{\mathcal{C}}$, $I \models \Delta_1$ iff $I \models \Delta_2$.

However, Lemma 5.5 does not extend to FO(ID) theories: for a definition Δ, $\mathcal{C}\langle\Delta\rangle$ is not necessarily $\mathcal{C}$-equivalent to Δ.

Example 5.6. Let σ be the empty vocabulary and T the theory

$$P,$$
$$\{P \leftarrow P\}.$$

This theory is unsatisfiable because the definition $\{P \leftarrow P\}$ has only one model, in which P is false. This contradicts the sentence in T. Clearly, $\top$ is a ct-bound for P. If $\mathcal{C}$ is a c-map for T over σ assigning $(\top, \bot)$ to P, then $\mathcal{C}\langle\{P \leftarrow P\}\rangle = \{P \leftarrow (P \wedge \neg\bot) \vee \top\}$, which is equivalent to $\{P \leftarrow \top\}$. This definition has only a model that assigns true to P. Since this model also satisfies $\mathcal{C}$, we conclude that $\{P \leftarrow P\}$ and $\mathcal{C}\langle\{P \leftarrow P\}\rangle$ are not $\mathcal{C}$-equivalent.

Definition 5.17. Let Δ a definition of T. We call c-map $\mathcal{C}$ for T *Δ-tolerant* if $\mathcal{C}\langle\Delta\rangle$ and Δ are $\mathcal{C}$-equivalent. We call $\mathcal{C}$ *T-tolerant* if it is Δ-tolerant for every definition Δ of T.

In the following, we say that a formula occurs positively (negatively) in a definition Δ if it occurs positively (negatively) in a body of a rule in Δ.

Proposition 5.13. *Let Δ be a definition of a theory T. Then a c-map $\mathcal{C}$ for T over σ is Δ-tolerant if for every subformula φ of Δ that contains a predicate $P \in \mathrm{Def}(\Delta)$, the following hold:*

1. *If Δ is not total, then $\mathcal{C}^{\mathrm{ctb}}(\varphi) = \mathcal{C}^{\mathrm{cfb}}(\varphi) = \bot$.*
2. *If φ occurs positively in Δ and P occurs positively in φ, then $\mathcal{C}^{\mathrm{ctb}}(\varphi) = \bot$.*
3. *If φ occurs negatively in Δ and P occurs negatively in φ, then $\mathcal{C}^{\mathrm{cfb}}(\varphi) = \bot$.*

Note that the c-map of Example 5.6 violates the second condition. The third condition is similar to the second one. The following example illustrates that also the first condition is needed.

Example 5.7. Let T be the theory

$$P, \tag{5.9}$$
$$\left\{ \begin{array}{ll} P & \leftarrow \neg Q \\ Q & \leftarrow \neg P \end{array} \right\}. \tag{5.10}$$

This theory is unsatisfiable, because definition (5.10) has a three-valued model. Since P occurs as a fact in T, assigning $(\top, \bot)$ to P and $(\bot, \bot)$ to every other subformula of T yields a c-map for T. The c-transformation of T with respect to this c-map is the theory

$$P \vee \top.$$
$$\left\{ \begin{array}{ll} P & \leftarrow \neg Q \\ Q & \leftarrow \neg(P \vee \top) \end{array} \right\}$$

This theory has a model where P is true and Q is false.

We will prove Proposition 5.13 by inductively constructing for any structure I that satisfies $\mathcal{C}$, a sequence of three-valued structures that is a well-founded induction extending I for both Δ and $\mathcal{C}\langle\Delta\rangle$. If $I \models \Delta$, we show that a terminal sequence with this property can be constructed, proving that I also satisfies $\mathcal{C}\langle\Delta\rangle$. If $I \not\models \Delta$, a sequence with this property can be constructed such that its last element is not less precise than I. This shows that I does not satisfy $\mathcal{C}\langle\Delta\rangle$ either. To construct a well-founded induction for both Δ and $\mathcal{C}\langle\Delta\rangle$, we prove that each step that extends a well-founded induction for Δ is also a valid step to extend it for $\mathcal{C}\langle\Delta\rangle$. Step (3a) in Definition 3.5 is covered by Lemma 5.14, step (3b) by Lemma 5.15.

Lemma 5.14. *Let I be a structure that satisfies a c-map $\mathcal{C}$ for T over σ and let $\tilde{J} \leq_p I$ be a three-valued interpretation such that $\tilde{J}|_\sigma$ is two-valued. Then $\tilde{J}\theta(\varphi) \leq_p \tilde{J}\theta(\mathcal{C}\langle\varphi\rangle)$ for every subformula φ of T and variable assignment θ.*

Proof. We prove this lemma by induction. First assume $\varphi[\overline{x}]$ is an atom. Then $\mathcal{C}\langle\varphi\rangle$ is the formula $(\varphi \wedge \neg\mathcal{C}^{\mathrm{cfb}}(\varphi)) \vee \mathcal{C}^{\mathrm{ctb}}(\varphi)$. If $\tilde{J}\theta(\varphi) = \mathbf{u}$, then clearly $\tilde{J}\theta(\mathcal{C}\langle\varphi\rangle) \geq_p \tilde{J}\theta(\varphi)$. If $\tilde{J}\theta(\varphi) = \mathbf{f}$, then $\tilde{J}\theta(\mathcal{C}^{\mathrm{ctb}}(\varphi))$ must be false, since $I \models \mathcal{C}$. Therefore $\tilde{J}\theta(\mathcal{C}\langle\varphi\rangle) = \mathbf{f}$. If on the other hand, $\tilde{J}\theta(\varphi) = \mathbf{t}$, then $\tilde{J}\theta(\mathcal{C}^{\mathrm{cfb}}(\varphi)) = \mathbf{f}$ and hence, $\tilde{J}\theta(\mathcal{C}\langle\varphi\rangle) = \mathbf{t}$.
The inductive cases are all very similar to the base case. We prove one of them. Assume φ is the formula $\psi \vee \chi$. Then $\mathcal{C}\langle\varphi\rangle$ is the formula $((\mathcal{C}\langle\psi\rangle \vee \mathcal{C}\langle\chi\rangle) \wedge \neg\mathcal{C}^{\mathrm{cfb}}(\varphi)) \vee \mathcal{C}^{\mathrm{ctb}}(\varphi)$. If $\tilde{J}\theta(\varphi) = \mathbf{f}$, then $\tilde{J}\theta(\psi) = \tilde{J}\theta(\chi) = \mathbf{f}$, and by induction $\tilde{J}\theta(\mathcal{C}\langle\psi\rangle) = \tilde{J}\theta(\mathcal{C}\langle\chi\rangle) = \mathbf{f}$. Since also $\tilde{J}\theta(\mathcal{C}^{\mathrm{ctb}}(\varphi)) = \mathbf{f}$, we conclude that $\tilde{J}\theta(\mathcal{C}\langle\varphi\rangle) = \mathbf{f}$. If on the other hand $\tilde{J}\theta(\varphi) = \mathbf{t}$, then $\tilde{J}\theta(\mathcal{C}^{\mathrm{cfb}}(\varphi)) = \mathbf{f}$. Also $\tilde{J}\theta(\psi) = \mathbf{t}$ or $\tilde{J}\theta(\chi) = \mathbf{t}$, and therefore $\tilde{J}\theta(\mathcal{C}\langle\psi\rangle) = \mathbf{t}$ or $\tilde{J}\theta(\mathcal{C}\langle\chi\rangle) = \mathbf{t}$. Hence $\tilde{J}\theta(\mathcal{C}\langle\varphi\rangle) = \mathbf{t}$. ■

Lemma 5.15. *Let Δ be a definition of T and $\mathcal{C}$ a c-map for T over σ that satisfies the three conditions of Proposition 5.13. Let I be a structure that satisfies $\mathcal{C}$ and $\tilde{J} \leq_p I$ a three-valued interpretation such that $\tilde{J}|_\sigma$ is two-valued. If U is a set of domain atoms defined in Δ and unknown in $\tilde{J}$, then for every variable assignment θ and subformula φ of Δ such that $\tilde{J}[U/\mathbf{f}]\theta(\varphi) \neq \mathbf{u}$, the following hold:*

- $\tilde{J}[U/\mathbf{f}]\theta(\varphi) \leq_t \tilde{J}[U/\mathbf{f}]\theta(\mathcal{C}\langle\varphi\rangle)$ *if φ occurs negatively in Δ;*

- $\tilde{J}[U/\mathbf{f}]\theta(\varphi) \geq_t \tilde{J}[U/\mathbf{f}]\theta(\mathcal{C}\langle\varphi\rangle)$ *if* φ *occurs positively in* Δ*;*

Proof. Let $\tilde{H}$ be the structure $\tilde{J}[U/\mathbf{f}]$. If $\tilde{J}\theta(\varphi) \neq \mathbf{u}$, the result follows immediately from Lemma 5.14.
We prove the case where $\tilde{J}\theta(\varphi) = \mathbf{u}$ by induction. Assume that φ is an atom $P(\overline{x})$. Since $\tilde{J}\theta(\varphi) = \mathbf{u}$ and $\tilde{H}\theta(\varphi) \neq \mathbf{u}$, we know that $P(\theta(\overline{x})) \in U$ and $\tilde{H}\theta(\varphi) = \mathbf{f}$. Therefore $\tilde{H}\theta(\mathcal{C}\langle\varphi\rangle) = \tilde{H}\theta((\varphi \wedge \neg\mathcal{C}^{\mathrm{cfb}}(\varphi)) \vee \mathcal{C}^{\mathrm{ctb}}(\varphi)) = \tilde{H}\theta(\mathcal{C}^{\mathrm{ctb}}(\varphi))$. If φ occurs negatively in Δ, then we have to prove that $\tilde{H}\theta(\varphi) \leq_t \tilde{H}\theta(\mathcal{C}\langle\varphi\rangle)$. Since $\tilde{H}\theta(\varphi) = \mathbf{f}$, this inequality holds regardless the value of $\mathcal{C}^{\mathrm{ctb}}(\varphi)$ and $\mathcal{C}^{\mathrm{cfb}}(\varphi)$ in $\tilde{H}\theta$. If on the other hand, φ occurs positively, we have to prove that $\tilde{H}\theta(\varphi) \geq \tilde{H}\theta(\mathcal{C}\langle\varphi\rangle)$. Since $\tilde{H}\theta(\varphi) = \mathbf{f}$ and $\tilde{H}\theta(\mathcal{C}\langle\varphi\rangle) = \tilde{H}\theta(\mathcal{C}^{\mathrm{ctb}}(\varphi))$, this inequality can only hold if $\tilde{H}\theta(\mathcal{C}^{\mathrm{ctb}}(\varphi)) = \mathbf{f}$. Because the conditions on $\mathcal{C}$ ensure that $\mathcal{C}^{\mathrm{ctb}}(\varphi) = \bot$, we can conclude that indeed $\tilde{H}\theta(\mathcal{C}^{\mathrm{ctb}}(\varphi)) = \mathbf{f}$.
We omit the inductive cases, since they are very similar to the base case. ■

Proof of Proposition 5.13. Let I be a structure that satisfies $\mathcal{C}$. We have to prove that $I \models \Delta$ iff $I \models \mathcal{C}\langle\Delta\rangle$. If Δ is not total, the proof is trivial, since then Δ and $\mathcal{C}\langle\Delta\rangle$ are equivalent.
Now assume that Δ is total and let $\langle\tilde{J}_\xi\rangle_{0\leq\xi\leq\alpha}$ be a well-founded induction for both Δ and $\mathcal{C}\langle\Delta\rangle$ above I. We will prove that if $\tilde{J}_\alpha$ is not two-valued, and $\tilde{J}_\alpha <_p I$, there exists a $\tilde{J}_{\alpha+1}$ such that $\langle\tilde{J}_\xi\rangle_{0\leq\xi\leq\alpha+1}$ is again a well-founded induction for Δ and $\mathcal{C}\langle\Delta\rangle$. Also observe that if λ is a limit ordinal and $\langle\tilde{J}_\xi\rangle_{0\leq\xi<\lambda}$ is a well-founded induction for both Δ and $\mathcal{C}\langle\Delta\rangle$, then the same holds for $\langle\tilde{J}_\xi\rangle_{0\leq\xi\leq\lambda}$. This is sufficient to conclude the proof. Indeed, if $I \models \Delta$, we can keep on extending the sequence until we end up in I, and derive that $I \models \mathcal{C}\langle\Delta\rangle$. If $I \not\models \Delta$, then we will eventually extend the well-founded induction with a structure $\tilde{J}_{\alpha+1} \not\leq_p I$. But then, the well-founded model of $\mathcal{C}\langle\Delta\rangle$ will also be more precise than $\tilde{J}_{\alpha+1}$, which shows that $I \not\models \mathcal{C}\langle\Delta\rangle$.
Assume that $\tilde{J}_\alpha$ is not two-valued and $\tilde{J}_\alpha <_p I$. Because Δ is total, there exists a $\tilde{J}_{\alpha+1}$ such that $\langle\tilde{J}_\xi\rangle_{0\leq\xi\leq\alpha+1}$ is a well-founded induction for Δ. We have to prove that it is also a well-founded induction for $\mathcal{C}\langle\Delta\rangle$. There are two possibilities:

- $\tilde{J}_{\alpha+1} = \tilde{J}_\alpha[V/\mathbf{t}]$ for some set V of domain atoms such that for every $P(\overline{d}) \in V$ there is a rule $\forall\overline{x}\ (P(\overline{x}) \leftarrow \varphi)$ in Δ such that $\tilde{J}_\alpha[\overline{x}/\overline{d}](\varphi) = \mathbf{t}$. By Lemma 5.14, also $\tilde{J}_\alpha[\overline{x}/\overline{d}](\mathcal{C}\langle\varphi\rangle) = \mathbf{t}$. Hence, $\langle\tilde{J}_\xi\rangle_{0\leq\xi\leq\alpha+1}$ is a well-founded induction for $\mathcal{C}\langle\Delta\rangle$.

- $\tilde{J}_{\alpha+1} = \tilde{J}_\alpha[U/\mathbf{f}]$ and for every $P(\overline{d}) \in U$ and rule $\forall\overline{x}\ (P(\overline{x}) \leftarrow \varphi)$ in Δ, $\tilde{J}_{\alpha+1}[\overline{x}/\overline{d}](\varphi) = \mathbf{f}$. From Lemma 5.15, it follows that $\tilde{J}_{\alpha+1}[\overline{x}/\overline{d}](\mathcal{C}\langle\varphi\rangle) = \mathbf{f}$. Therefore, $\langle\tilde{J}_\xi\rangle_{0\leq\xi\leq\alpha+1}$ is a well-founded induction for $\mathcal{C}\langle\Delta\rangle$.

■

From Proposition 5.13 we derive the following procedure to compute a T-tolerant c-map for a theory T. First compute a c-map $\mathcal{C}$ for T that is not necessarily T-tolerant. Then, for every definition Δ of T and every subformula φ of Δ,

replace $\mathcal{C}^{\text{ctb}}(\varphi)$ and $\mathcal{C}^{\text{cfb}}(\varphi)$ by $\bot$, if this is required to satisfy the conditions of Proposition 5.13. It can easily be proven by induction that if the original c-map $\mathcal{C}$ was atom-based, the obtained c-map is still atom-based. On the other hand, it is not necessarily the case that the obtained c-map is atom-equal if the original one was.
We conclude that the following algorithm produces a correct grounding for FO(ID) theory T:

1. Compute an atom-based c-map $\mathcal{C}$ for T over σ.
2. If $\mathcal{C}$ is inconsistent with respect to I_σ, output $\bot$ and stop.
3. Else, derive an atom-based, T-tolerant c-map $\mathcal{C}'$ from $\mathcal{C}$.
4. Output $\text{Gr}_{\text{red}}(\mathcal{C}'\langle T\rangle \cup \overline{\mathcal{C}'}_A)$, using any off-the-shelf grounder for FO(ID).

5.3.2 Grounding aggregates with bounds

Similarly as for FO and FO(ID), every FO(AGG) and FO($\cdot$) model expansion problem can be grounded to an equivalent satisfiability problem in the propositional fragment of, respectively, FO(AGG) and FO($\cdot$). Such satisfiability problems can be solved by the systems MIDL (if all aggregates are over the aggregate function CARD) and MINISAT(ID). Mariën (2009) describes details of the specific form of propositional FO($\cdot$) theories accepted by MINISAT(ID). Below, we define full and reduced grounding for sentences of the form $\text{F}(V) \leq d$, $\text{F}(V) \geq d$ and $\text{F}V = d$. Adding these to Definition 5.4 provides the definition of full and reduced grounding for FO(AGG) and FO($\cdot$) theories in TNF.

Definition 5.18. The *full grounding* of a set expression $\{(x, \overline{y}) \mid \varphi[x, \overline{y}]\}$ with respect to finite domain D is the set $\{(d_x, \text{Gr}_{\text{full}}(\varphi[x/d_x, \overline{y}/\overline{d}_y], D)) \mid d_x \in D \text{ and } \overline{d}_y \in D^{|y|}\}$. The full grounding of the set expression $\{(d_1, \varphi_1), \ldots, (d_n, \varphi_n)\}$ is the set $\{(d_1, \text{Gr}_{\text{full}}(\varphi_1, D)), \ldots, (d_n, \text{Gr}_{\text{full}}(\varphi_1, D))\}$. We denote the full grounding of a set expression V by $\text{Gr}_{\text{full}}(V, D)$. The full grounding of sentences of the form $(\text{F}(V) \leq d)$, $(\text{F}(V) \geq d)$ and $(\text{F}(V) = d)$ are, respectively, the sentences $(\text{F}(\text{Gr}_{\text{full}}(V, D)) \leq d)$, $(\text{F}(\text{Gr}_{\text{full}}(V, D)) \geq d)$ and $(\text{F}(\text{Gr}_{\text{full}}(V, D)) = d)$

The reduced grounding of a FO($\cdot$) theory T with respect to I_σ is obtained by replacing in $\text{Gr}_{\text{full}}(T, D)$ all subformulas φ over σ by $\top$ if $I_\sigma \models \varphi$ and by $\bot$ otherwise. The resulting theory may be simplified by replacing, e.g., the formula

$$\text{SUM}\{(m_1, \top), (m_2, \bot), (m_3, \varphi_3), \ldots, (m_n, \varphi_n)\} \leq m$$

by

$$\text{SUM}\{(m_3, \varphi_3), \ldots, (m_n, \varphi_n)\} \leq m - m_1,$$

the formula $\text{CARD}\{\varphi_1, \ldots, \varphi_n\} \geq 0$ by $\top$, etc.
All results about grounding with bounds for FO theories extend to FO(AGG) theories, since the result of Lemma 5.5 holds also for FO(AGG) theories. To extend our results to FO($\cdot$), we need to find conditions that guarantee Δ-tolerance

monotone	anti-monotone
$\text{CARD}(V) \geq z$	$\text{CARD}(V) \leq z$
$\text{MAX}(V) \geq z$	$\text{MAX}(V) \leq z$
$\text{MIN}(V) \leq z$	$\text{MIN}(V) \geq z$
$\text{SUM}\{(x, \overline{y}) \mid x \geq 0 \wedge \varphi\} \geq z$	$\text{SUM}\{(x, \overline{y}) \mid x \geq 0 \wedge \varphi\} \leq z$
$\text{SUM}\{(x, \overline{y}) \mid x \leq 0 \wedge \varphi\} \leq z$	$\text{SUM}\{(x, \overline{y}) \mid x \leq 0 \wedge \varphi\} \geq z$
$\text{PROD}\{(x, \overline{y}) \mid x > 0 \wedge \varphi\} \geq z$	$\text{PROD}\{(x, \overline{y}) \mid x > 0 \wedge \varphi\} \leq z$

Table 5.1: Monotone and anti-monotone aggregate expressions.

of a c-map in the case Δ contains aggregate expressions. These conditions should ensure a lemma similar to Lemma 5.15 can be proven for definitions containing aggregates. Intuitively, this lemma states that the truth value of a rule body in a definition does not increase (according to the truth order) when bounds are added. Before we state sufficient conditions to ensure this property, we introduce the notions of monotone and anti-monotone aggregate expressions.

Definition 5.19. An aggregate expression φ is *monotone* if $\tilde{I}(\varphi) \leq_t \tilde{J}(\varphi)$ for any two structures $\tilde{I}$ and $\tilde{J}$ such that $\tilde{I}(\psi) \leq_t \tilde{J}(\psi)$ for every direct subformula ψ of φ. An aggregate expression φ is *anti-monotone* if $\tilde{I}(\varphi) \geq_t \tilde{J}(\varphi)$ for any two structures $\tilde{I}$ and $\tilde{J}$ such that $\tilde{I}(\psi) \leq_t \tilde{J}(\psi)$ for every direct subformula ψ of φ.

Table 5.1 lists some monotone and anti-monotone aggregate expressions.
The following proposition lists conditions on a c-map that ensure Δ-tolerance for definitions that may contain aggregate expressions.

Proposition 5.16. *Let Δ be a definition of a* FO($\cdot$) *theory T. Then a c-map $\mathcal{C}$ for T over σ is Δ-tolerant if for every subformula φ of Δ that contains a predicate $P \in \text{Def}(\Delta)$, the following hold:*

- *If Δ is not total or φ occurs in an aggregate expression that is neither monotone nor anti-monotone, then $\mathcal{C}^{\text{ctb}}(\varphi) = \mathcal{C}^{\text{cfb}}(\varphi) = \bot$;*
- *If φ occurs in the scope of an even number of negations and anti-monotone aggregate expressions and P occurs in the scope of an even number of negations and anti-monotone aggregates in φ, then $\mathcal{C}^{\text{ctb}}(\varphi) = \bot$.*
- *If φ occurs in the scope of an odd number of negations and anti-monotone aggregate expressions and P occurs in the scope of an odd number of negations and anti-monotone aggregates in φ, then $\mathcal{C}^{\text{cfb}}(\varphi) = \bot$.*

Proof. The proof is analogous to the proof of Proposition 5.13. ■

5.4 Implementation and experiments

So far we have focussed mostly on grounding size. Proposition 5.11 guaranteed that grounding with bounds produces smaller groundings for FO theories. In

this section we are concerned with the efficiency and practical implementation of grounding with bounds. A first issue was mentioned in Example 5.5: an atom-based c-map $\mathcal{C}$ corresponding to a symbolic structure contains many repeated constraints on variables. To ground $\mathcal{C}\langle T\rangle$ efficiently, such repetitions should be avoided as much as possible. Secondly, an efficient grounder consults bounds as soon as possible. In particular, it should use bounds to avoid unnecessary instantiations of variables, rather than to remove these instantiations afterwards. As a case study, we will show in detail how to adapt a basic "top-down style" grounding algorithm to efficiently exploit bounds. We sketch how the same principles can be applied for a "bottom-up style" grounder.

In the second part of this section we discuss a stop criterion for the symbolic propagation algorithm, i.e., an implementation of the function `acceptable` in Algorithm 4.3. Finally, we report on our implementation, called GIDL, of the symbolic propagation and grounding algorithm. We present experimental results that show the impact of using bounds on grounding size and time.

5.4.1 Case study: top-down grounding with bounds

For the rest of this section, assume T is in TNF and fix an I_σ-consistent, atom-based, T-tolerant c-map $\mathcal{C}$ for T over σ. We call a formula of the form $(\varphi \vee \psi)$ or $(\exists x\ \varphi)$ a *disjunctive formula*. Vice versa, a *conjunctive formula* is a formula of the form $(\varphi \wedge \psi)$ or $(\forall x\ \varphi)$.

We now present a simple "top-down style" grounding algorithm that exploits bounds without constructing $\mathcal{C}\langle T\rangle$ explicitly. The algorithm is shown in Algorithm 5.1. Basically, it consults the bounds assigned by $\mathcal{C}$ whenever it substitutes the free variables of a formula $\varphi[\overline{x}]$ by domain constants $\overline{d}$. If according to the bounds, $\varphi[\overline{x}/\overline{d}]$ is certainly true, i.e., $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}^{\mathrm{ctb}}(\varphi)$, then the grounding of $\varphi[\overline{x}/\overline{d}]$ is not computed. Instead, the algorithm then proceeds as if $\varphi[\overline{x}/\overline{d}]$ is equal to $\top$. Similarly if $\varphi[\overline{x}/\overline{d}]$ is certainly false. In this way, the algorithm avoids creating unnecessary instantiations. One can check that if $\mathcal{C}$ is the trivial c-map, Algorithm 5.1 reduces to a straightforward top-down style grounding algorithm that produces $\mathrm{Gr}_{\mathrm{full}}(T)$.

Line 1 of Algorithm 5.1 checks whether one of the sentences of T is certainly false. If this is the case, then clearly T is unsatisfiable (cf. Definition 5.9), and this can be reported immediately. Before a sentence is grounded, line 4 checks whether this sentence is certainly true according to $\mathcal{C}$. Only sentences that are not certainly true are grounded. Observe that both checks are simple syntactic checks and can be executed in constant time.

Function `groundConj` gets as input a formula $\varphi[\overline{x}]$ and returns a grounding for $(\forall\overline{x}\ \varphi[\overline{x}])$. In particular, if φ is a sentence, then the result of applying `groundConj` to φ is a grounding for φ.

In `groundConj`, universal quantifiers are implicitly pushed inside conjunctions. That is, if $\varphi[\overline{x}]$ is a conjunction $\psi_1 \wedge \ldots \wedge \psi_n$, then for every $i \in [1, n]$, the grounding of $(\forall\overline{x}\ \psi_i)$ is computed by applying `groundConj` to ψ_i. The conjunction of these groundings is returned as grounding for $(\forall\overline{x}\ \varphi)$. According

Algorithm 5.1: Ground with bounds

Input: T, σ, I_σ and $\mathcal{C}$
Output: A grounding T_g for T with respect to I_σ

if $\mathcal{C}^{\mathrm{cfb}}(\varphi) = \top$ *for some sentence* φ *of* T **then return** $\bot$;
$T_\mathrm{g} := \emptyset$;
`// Ground all sentences of` T
for *every sentence* φ *of* T **do**
  if $\mathcal{C}^{\mathrm{ctb}}(\varphi) \neq \top$ **then** Add `groundConj(`φ`)` to T_g;
`// Ground all definitions of` T
for *every definition* Δ *of* T **do**
  Add `groundDef(`Δ`)` to T_g;
`// Add the grounding of` $\overline{\mathcal{C}}_A$
for *every atomic subformula* $\varphi[\overline{x}]$ *of* T **do**
  for *every* $\overline{d}$ *such that* $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}^{\mathrm{ctb}}(\varphi)$ **do**
    Add $\varphi[\overline{x}/\overline{d}]$ to T_g;
  for *every* $\overline{d}$ *such that* $I_\sigma[\overline{x}/\overline{d}] \models \mathcal{C}^{\mathrm{cfb}}(\varphi)$ **do**
    Add $\neg\varphi[\overline{x}/\overline{d}]$ to T_g;
return T_g;

Function `groundConj(`$\varphi[\overline{x}]$`)`

$C := \emptyset$;
switch $\varphi[\overline{x}]$ **do**
  case $\varphi = \forall y\ \psi[\overline{x}, y]$
    return `groundConj(`$\psi[\overline{x}, y]$`)`;
  case $\varphi = \bigwedge_i \psi_i$
    $C := \bigcup_i$ `groundConj(`ψ_i`)`;
  otherwise
    for *all* $\overline{d}$ *such that* $I_\sigma \not\models \mathcal{C}^{\mathrm{ctb}}(\varphi)[\overline{x}/\overline{d}]$ **do**
      if $I_\sigma \models \mathcal{C}^{\mathrm{cfb}}(\varphi)[\overline{x}/\overline{d}]$ **then return** $\bot$;
      else
        if φ *is a literal* **then**
          Add $\varphi[\overline{x}/\overline{d}]$ to C;
        if φ *is a disjunctive formula* **then**
          Add `groundDisj(`$\varphi[\overline{x}/\overline{d}]$`)` to C;
        if φ *is an aggregate expression* **then**
          Add `groundAgg(`$\varphi[\overline{x}/\overline{d}]$`)` to C;
return $\bigwedge C$;

Function `groundDisj(`$\varphi[\overline{x}]$`)`

```
1  D := ∅;
2  switch φ[x̄] do
3      case φ = ∃y ψ[x̄, y]
4          return groundDisj(ψ[x̄, y]);
5      case φ = ⋁_i ψ_i
6          D := ⋃_i groundDisj(ψ_i);
7      otherwise
8          for all d̄ such that I_σ ⊭ C^cfb(φ)[x̄/d̄] do
9              if I_σ ⊨ C^ctb(φ)[x̄/d̄] then return ⊤;
10             else
11                 if φ is a literal then
12                     Add φ[x̄/d̄] to D;
13                 if φ is a conjunctive formula then
14                     Add groundDisj(φ[x̄/d̄]) to D;
15                 if φ is an aggregate expression then
16                     Add groundAgg(φ[x̄/d̄]) to D;
17 return ⋁ D;
```

Function `groundDef(`Δ`)`

```
1  Δ_g := ∅;
2  for every rule ∀x̄ (P(x̄) ← φ[ȳ]) in Δ do
3      z̄ := x̄ \ ȳ;
4      for every d̄ such that I_σ ⊭ C^cfb(φ[ȳ/d̄]) do
5          if I_σ ⊨ C^ctb(φ[ȳ/d̄]) then φ_g := ⊤;
6          else φ_g := groundConj(φ[ȳ/d̄]);
7          Add P(x̄)[ȳ/d̄, z̄/d̄'] ← φ_g to Δ_g for every d̄' ∈ D^|z̄|;
8  return Δ_g;
```

Function `groundAgg(`φ`)`

```
1 switch φ do
2     case φ = CARD{x̄ | ψ} ≥ n
3         V := ∅;
4         for every d̄ such that I_σ[x̄/d̄] ⊭ C^cfb(ψ) do
5             if I_σ[x̄/d̄] ⊨ C^ctb(ψ) then n := n − 1;
6             else Add groundConj(ψ[x̄/d̄]) to V;
7         if n ≤ 0 then return ⊤;
8         else return CARD(V) ≥ n;
9     ⋮    // Similar code for other aggregate expressions
```

to equivalence (2.5) of Section 2.1.3, this transformation yields an equivalent formula.

Function `groundConj` only consults the c-map when variables are substituted by domain constants or when the input formula is an atom. As such, `groundConj` ignores ("eliminates") the bounds assigned to conjunctive formulas. As we mentioned in Example 5.5, this is important to avoid repeated constraints on a variable.

In `groundConj(`$\varphi[\overline{x}]$`)`, only those substitutions $\varphi[\overline{x}/\overline{d}]$ for which $I_\sigma[\overline{x}/\overline{d}] \not\models \mathcal{C}^{\text{ctb}}(\varphi)$ are grounded (line 8). Indeed, the other substitutions yield a formula that is certainly true in all models of T expanding I_σ, and can therefore be omitted from the ground conjunction C that is computed. Before $\varphi[\overline{x}/\overline{d}]$ is grounded, it is checked whether this substitution yields a formula that is certainly false (line 9). If this is the case, the whole conjunction C will certainly be false, and therefore $\bot$ is returned immediately. Observe that *implicitly* the formula $\mathcal{C}^{\text{ctb}}(\varphi) \vee (\neg\mathcal{C}^{\text{cfb}}(\varphi) \wedge \varphi)$ is grounded. Hence the correctness of `groundConj` follows from Lemma 5.4.

Function `groundDisj` is dual to `groundConj`. On input $\varphi[\overline{x}]$, it returns a grounding for $\exists\overline{x}\ \varphi[\overline{x}]$. It implicitly pushes existential quantifiers through disjunctions and eliminates the bounds assigned to disjunctive formulas.

Function `groundDef` returns a grounding for its input definition Δ. It grounds the rules of Δ one-by-one. For each rule $\forall\overline{x}\ (P(\overline{x}) \leftarrow \varphi[\overline{y}])$, only those substitutions $\varphi[\overline{y}/\overline{d}]$ that are possibly true are tried (line 4). If $\varphi[\overline{y}/\overline{d}]$ is certainly true, it is replaced by $\top$ (line 5). Function `groundAgg` returns a grounding for aggregate expressions.

The computationally expensive steps in Algorithm 5.1 are the steps where the truth values in I_σ of (some of the) bounds assigned by $\mathcal{C}$ are computed. For large bounds, these steps can become infeasible. Indeed, the expression complexity of FO is **PSPACE**-complete (Theorem 2.1). As such, grounding with too complex bounds may take more time and space than constructing the full grounding and simplifying it afterwards. The stop criterion of Section 5.4.2 for the symbolic propagation algorithm is designed to avoid too complex bounds.

Our experiments in Section 5.4.3 show that carefully restricting the complexity of the bounds leads to faster grounding.
We stress that Algorithm 5.1 is just one example of a grounding algorithm that exploits bounds.[23] The principle of consulting bounds as soon as possible can be applied to adapt other grounding algorithms as well. For example, recall that a bottom-up style grounder starts by storing all instances of atomic subformulas of T in a table. To exploit bounds efficiently, a bottom-up grounder should consult the bounds while constructing these tables and leave out, e.g., all instances that are certainly false. As such, it avoids unnecessary large tables, which in turn improves the speed of the subsequent grounding steps.

5.4.2 A stop criterion for symbolic propagation

As shown in Section 4.3.2, the symbolic propagation algorithm does not reach a fixpoint on certain inputs. Also, even in the case a fixpoint can be found, computing it may take a long time, and the resulting c-map can be so complex that querying becomes very inefficient. Using such a c-map may severely slow down grounding. This indicates the need for a stop criterion that avoids too complicated c-maps. As such, the function `acceptable` in Algorithm 4.3 should only accept not too complicated formulas.
If Algorithm 4.3 is implemented using BDDs, a simple way to limit the complexity of formulas is by putting a fixed upper bound N on the number of nodes the BDD representation of a bound may have. That is, $\texttt{acceptable}(\varphi,\_)$ is true iff the BDD representation of φ has less than N nodes. The experiments we present in Section 5.4.3 indicate that there exist appropriate values for the maximum number of iterations of the while loop in Algorithm 4.3 (i.e., the constant C) and N that produce positive results on most of the examples. Still, on some problems, grounding slows down severely, while the size of the produced grounding does not decrease. One of these problems is the following *clique problem* (entry 6 in Table 5.3).

Example 5.8. Recall that a clique is a maximally connected graph. Let

$$\sigma = \langle \{Vtx\}, \{Edge(Vtx, Vtx)\}, \emptyset \rangle,$$
$$\Sigma = \langle \sigma_{\text{sort}}, \sigma_{\text{pred}} \cup \{Clique(Vtx)\}, \emptyset \rangle$$

and T the theory

$$\forall x \forall y\ (Clique(x) \wedge Clique(y) \Rightarrow (x = y \vee Edge(x, y))).$$
$$\forall x\ ((\forall y\ (Clique(y) \wedge x \neq y \Rightarrow Edge(x, y))) \Rightarrow Clique(x)).$$

If $Edge^{I_\sigma}$ is symmetric, i.e., I_σ represents an undirected graph, a model of T expanding I_σ is a clique in I_σ that is not contained in a strictly larger clique in I_σ. Within a small number of iterations, Algorithm 4.3 finds for $Clique(x)$ the

[23] The question whether top-down grounders can be made more efficient than bottom-up grounders is outside the scope of this thesis, and still undecided.

ct-bound $\forall x'\ (x \neq x' \Rightarrow \mathit{Edge}(x, x'))$. This formula expresses that $\mathit{Clique}(x)$ is certainly true in every solution if x is directly connected to every other vertex in the input graph. Clearly, for most graphs, no vertex satisfies this condition. So, for most graphs, $\bot$ would be an equally precise ct-bound, but would allow much faster querying.
The situation is worse for the cf-bound for $\mathit{Clique}(x)$. Since for an undirected graph, every single vertex is a clique, and thus occurs in at least one of the solutions, the cf-bound is necessarily unsatisfiable with respect to T. Yet, our implementation of the symbolic propagation algorithm came up with

$$\exists x'\ (\neg \mathit{Edge}(x, x') \wedge x \neq x' \wedge (\forall x''\ (x' \neq x'' \Rightarrow \mathit{Edge}(x', x''))))$$

as cf-bound. The query algorithm presented in Section 4.3.3 takes cubic time in the number of vertices to find out that no x satisfies this formula.

To avoid the problems illustrated by the example above, one could estimate the reward of a bound versus the cost of evaluating it. Recall that more precise bounds yield smaller grounding sizes. Therefore, the reward of a bound ψ is dictated by its precision. Given I_σ, it is possible to find a good estimate for the number of answers to ψ in I_σ (Demolombe, 1980), which is in turn a measure for the precision of ψ. For a fixed query algorithm, one can also estimate the cost $\mathrm{cost}(\psi)$ of computing an answer in I_σ to a query ψ. In the following, we assume that the reward of a bound is a positive real number, and its cost a strictly positive real number.
Given the reward and the cost of bounds, the complexity of a bound ψ can be limited by restricting the ratio

$$r(\psi) := \frac{\mathrm{cost}(\psi)}{\mathrm{reward}(\psi) + 1}.$$

If a propagator would replace a bound ψ_1 by ψ_2, but $r(\psi_1) < r(\psi_2)$, then this propagator is not applied. That is, `acceptable`(ψ_2,ψ_1) is only true if $r(\psi_2) < r(\psi_1)$. Clearly, for all bounds ψ assigned by a c-map $\mathcal{C}$ computed according to this restriction, $r(\psi) \leq r(\bot)$ holds. Observe that to apply this restriction, an input structure I_σ is needed. However, the obtained c-map is independent of I_σ.
The (fairly naive) estimators we implemented are described in detail in Appendix C. They assigns ratios of the order $\mathcal{O}(|Vtx^{I_\sigma}|)$, respectively $\mathcal{O}(|Vtx^{I_\sigma}|^3)$, to the ct-bound, respectively cf-bound, mentioned in Example 5.8. As such, if $|Vtx^{I_\sigma}|$ is large enough, these bounds will be avoided.

5.4.3 Experiments

We implemented Algorithm 5.1 and Algorithm 4.3, using BDDs to represent bounds. The resulting grounder is called GIDL. In this section, we present experiments, obtained with GIDL, that show the impact of using bounds on grounding size and time. Our implementation of Algorithm 4.3 does not yet

contain propagators involving aggregates or definitions. Propagation for definitions is done on the completion.
As input for GIDL, we used 37 benchmark problems, mainly taken from Asparagus.[24] Details about the experiments are available at `www.cs.kuleuven.be/~dtai/krr/software.html`. We used four different versions of GIDL:

GIDL$_{\text{nb}}$: Assigns $(\varphi, \neg\varphi)$ as bound to every atomic subformula φ over the input vocabulary, and $(\bot, \bot)$ to every other subformula. As such, it creates the reduced grounding of the input theory.

GIDL$_{\text{bu}}$: Assigns $\tilde{\Phi}_\sigma(\varphi)$ as bound to every subformula φ of T, where $\tilde{\Phi}_\sigma$ is the symbolic structure defined in Section 5.2.4.

GIDL$_{\text{mn}}$: Computes a c-map by applying symbolic propagation as explained in Section 5.2.4. The constant C in Algorithm 4.3 is set to four times the number of subformulas in T and a maximum of four internal nodes is allowed in each BDD used to represent the bounds. That is, the function `acceptable`(φ,_) returns true iff the BDD representation of φ has at most four internal nodes. According to previous experiments (Wittocx et al., 2008b), this is the best setting when limiting the number of nodes.

GIDL$_{\text{r}}$: Like GIDL$_{\text{mn}}$, but the complexity of the derived c-map is limited by estimating the number of answers and the cost, as described in Section 5.4.2.

In Table 5.2, the influence of bounds on the grounding size is shown. The second and third column show the ratio of the grounding size obtained with GIDL$_{\text{mn}}$ and GIDL$_{\text{r}}$ compared to $\text{Gr}_{\text{red}}(T)$. For GIDL$_{\text{nb}}$ and GIDL$_{\text{bu}}$, this ratio is always equal to 1. When interpreting Table 5.2, it is important to note that small reductions in grounding size are not important. The reason being that all reductions that can be obtained by symbolic propagation are also obtained by applying unit propagation on the grounding (see Appendix B). Since there exist very efficient implementations of unit propagation, it is not beneficial to let symbolic propagation find small reductions at a relatively high cost. In around 30% of the benchmarks, the size of the grounding produced by GIDL$_{\text{mn}}$ and GIDL$_{\text{r}}$ is less than half the size of the grounding computed by GIDL$_{\text{nb}}$. In 7, respectively 6, of the benchmarks the grounding size is even less than 5% of the size of GIDL$_{\text{nb}}$'s grounding.
More important than reductions in size are reductions in grounding time. Table 5.3 shows the running times of the different versions of GIDL, and (between brackets) the ratio of the running time to the running time of GIDL$_{\text{nb}}$. The running time of Algorithm 4.3 is included (it never took more than 0.02 seconds). A time-out (###) of 600 seconds was used.
On many benchmarks, the reduction in grounding time with respect to GIDL$_{\text{nb}}$ is due to the reduction in grounding size. Yet there are also several benchmarks where time decreases a lot, while there is almost no reduction in size. As can be seen from the running times of GIDL$_{\text{bu}}$, this is mostly due to the assignment

[24] `http://asp.haiti.cs.uni-potsdam.de/`

of non-trivial bounds to non-atomic subformulas. Non-trivial bounds for non-atomic subformulas allow for earlier pruning by a top-down style grounder, and hence faster grounding.
From Table 5.3, we can see that GIDL$_{\text{mn}}$ performs quite well. On half of the benchmarks, it is more than 44% faster than GIDL$_{\text{nb}}$. It is also more than 20% faster than GIDL$_{\text{bu}}$ on half of the benchmarks. There are some outliers however. On benchmarks 6 and 11, it is far slower than GIDL$_{\text{bu}}$, while not producing a significantly smaller grounding. This indicates the use of a complex bound with relatively small reward. Compared to GIDL$_{\text{mn}}$, GIDL$_{\text{r}}$ is faster and more robust, indicating that using estimators for the reward and cost of bounds pays off in most cases. In only two of the benchmarks, our naive estimator makes a wrong guess. In benchmark 1, a bound with high cost and no reward is allowed, in benchmark 7, a bound with low cost and high reward is not allowed by GIDL$_{\text{r}}$. It is part of future work to implement improved estimators.
We conclude from our experiments that grounding with bounds is applicable in practice. It often leads to smaller grounding sizes on standard benchmark problems, and if the bounds are carefully restricted, it yields a significant speed up. Since the time to compute bounds is small compared to the overall grounding time, computing them is essentially for free.
In general, a smaller grounding does not necessarily lead to faster propositional model generation. For example, grounding size (and time) increases when symmetry breaking formulas are added, but these formulas may drastically improve the overall solving time (Torlak and Jackson, 2007). Another example are clause-learning SAT solvers: the clauses learnt by these solvers are redundant, but may improve the solving time by orders of magnitude. The question arises whether our method of grounding with bounds may lead to slower overall model generation time compared to grounding without bounds. This is not the case. The experiments above show that in general, grounding with bounds is faster than grounding without bounds. Since grounding with bounds also produces smaller groundings, the subsequent initialization phase of the SAT solver is executed faster. If T_1 and T_2 are two groundings obtained by grounding the same input theory and structure with, respectively without bounds, results in Appendix B shows that the typical simplification steps applied in this initialization phase transform T_1 and T_2 in similar[25] theories. It follows that in general, the overall model generation time does not increase when bounds are applied while grounding.

5.4.4 The IDP system

GIDL can be used as a preprocessor for the propositional solvers MIDL (Mariën et al., 2007a,b) and MINISAT(ID) (Mariën et al., 2008; Mariën, 2009). The resulting system is a finite model expander for FO($\cdot$), called IDP. The input language for GIDL, and hence for IDP, is an ASCII version of FO($\cdot$). An example input for IDP is given in Appendix C.

[25]The only difference is that the second theory may contain some extra auxiliary symbols introduced by the Tseitin transformation.

Nr	Benchmark name	GIDL$_{mn}$	GIDL$_{r}$
1	15puzzle	1.00	1.00
2	Battleship	0.89	1.00
3	Blocked N-queens	0.02	0.02
4	Blocksworld	0.33	0.33
5	Bounded spanningtree	0.12	0.12
6	Clique	1.00	1.00
7	Hierarchical clustering	0.03	0.72
8	Graph colouring	1.00	1.00
9	Debugging	0.86	1.00
10	Fastfood	1.00	1.00
11	FO-hamcircuit	0.94	0.99
12	Golomb ruler	0.54	1.00
13	Graph partitioning	0.94	1.00
14	Algebraic groups	0.99	1.00
15	Hamiltonian circuit	0.01	0.01
16	Tower of Hanoi	1.00	1.00
17	Knighttour	0.00	0.00
18	Labyrinth	0.99	0.99
19	Magic series	1.00	1.00
20	Maze generation	0.90	0.90
21	Mirror puzzle	1.00	1.00
22	Missionaries	0.03	0.03
23	N-queens	1.00	1.00
24	Pigeonhole	1.00	1.00
25	Disjunctive scheduling	0.83	0.83
26	Slitherlink	0.04	0.04
27	Social golfer	1.00	1.00
28	Sokoban	0.59	0.59
29	Solitaire	1.00	0.73
30	Spanningtree	0.06	0.06
31	Sudoku	0.75	0.75
32	Tarski	1.00	1.00
33	Toughnut	0.00	0.00
34	Train scheduling	0.25	0.25
35	Waterbucket	0.36	0.36
36	Weight bounded dominating set	1.00	1.00
37	Wire routing	0.92	0.99
	Average	0.66	0.70
	# < 1.00	24	20
	# < 0.50	12	11
	# < 0.05	7	6

Table 5.2: Impact of bounds on grounding size. The numbers are the ratio of the grounding size to the size of the reduced grounding if no bounds are used.

Nr	GIDL$_{nb}$	GIDL$_{bu}$		GIDL$_{mn}$		GIDL$_{r}$	
1	6.13	2.00	(0.33)	2.07	(0.34)	5.73	(0.93)
2	0.19	0.18	(0.95)	0.16	(0.84)	0.17	(0.89)
3	9.66	10.83	(1.12)	2.22	(0.23)	2.67	(0.28)
4	22.33	16.76	(0.75)	5.80	(0.26)	5.80	(0.26)
5	8.52	8.52	(1.00)	3.01	(0.35)	1.16	(0.14)
6	3.13	3.73	(1.19)	51.77	(16.54)	3.73	(1.19)
7	0.32	0.34	(1.06)	0.05	(0.16)	0.31	(0.97)
8	2.57	2.71	(1.05)	2.69	(1.05)	2.72	(1.06)
9	0.30	0.30	(1.00)	0.48	(1.60)	0.47	(1.57)
10	###	###	(1.00)	17.59	(0.03)	16.52	(0.03)
11	###	5.87	(0.01)	37.86	(0.06)	6.06	(0.01)
12	14.05	3.54	(0.25)	4.13	(0.29)	3.40	(0.24)
13	0.03	0.04	(1.33)	0.03	(1.00)	0.02	(0.67)
14	9.68	9.58	(0.99)	11.20	(1.16)	9.60	(0.99)
15	70.75	71.50	(1.01)	2.56	(0.04)	1.81	(0.03)
16	2.32	1.83	(0.79)	1.96	(0.84)	1.83	(0.79)
17	12.22	10.35	(0.85)	0.06	(0.00)	0.10	(0.01)
18	8.80	8.83	(1.00)	8.83	(1.00)	8.73	(0.99)
19	1.83	1.76	(0.96)	1.79	(0.98)	1.81	(0.99)
20	2.77	2.80	(1.01)	0.51	(0.18)	0.17	(0.06)
21	0.12	0.11	(0.92)	0.12	(1.00)	0.10	(0.83)
22	17.4	18.08	(1.04)	2.29	(0.13)	2.68	(0.15)
23	4.62	4.60	(1.00)	4.62	(1.00)	4.64	(1.00)
24	4.92	5.01	(1.02)	4.90	(1.00)	4.90	(1.00)
25	151.15	151.66	(1.00)	172.50	(1.14)	171.54	(1.13)
26	0.25	0.13	(0.52)	0.02	(0.08)	0.02	(0.08)
27	5.47	5.47	(1.00)	5.37	(0.98)	5.41	(0.99)
28	2.78	2.66	(0.96)	1.57	(0.56)	1.54	(0.55)
29	0.43	0.43	(1.00)	0.46	(1.07)	0.49	(1.14)
30	6.86	6.79	(0.99)	0.59	(0.09)	0.57	(0.08)
31	###	2.34	(0.00)	1.07	(0.00)	1.06	(0.00)
32	4.42	4.53	(1.02)	3.67	(0.83)	3.64	(0.82)
33	4.23	4.23	(1.00)	0.53	(0.13)	0.53	(0.13)
34	4.06	2.14	(0.53)	0.65	(0.16)	0.47	(0.12)
35	3.16	3.07	(0.97)	1.76	(0.56)	2.04	(0.65)
36	1.45	1.42	(0.98)	0.03	(0.02)	0.03	(0.02)
37	0.06	0.06	(1.00)	0.08	(1.33)	0.08	(1.33)
Total	2186.98	974.20	(0.45)	355.00	(0.16)	272.55	(0.12)
Avg. gain			12 %		0 %		40%
Median gain			0 %		44 %		33%

Table 5.3: Impact of bounds on grounding time. All times are in seconds. For GIDL$_{mn}$ and GIDL$_{r}$, the time to compute the bounds is included. The numbers between brackets are the relative times with respect to GIDL$_{nb}$.

IDP performed quite well in the latest Answer Set Programming Competition (Denecker et al., 2009). In the category *Decision problems in* **NP** , it ranked third among the fourteen *single system* teams, i.e., among the teams that applied the same system for each of the problems. IDP is used in LogicPalet[26], a software tool to help students master the basic concepts of mathematical logic.

5.5 Related work

In the previous sections we described a method to obtain fast and compact grounding. Several such methods have been described in the literature. Some of them are — like ours — preprocessing techniques that rewrite the input theory. Other techniques involve reasoning on the propositional level. In this section we provide an overview. We indicate which ones can be applied to improve GIDL. We also give an overview of existing grounders.

5.5.1 Methods to optimize grounding

Derivation of Bounds To our knowledge, the methods proposed in the literature to derive bounds are less general than the one we presented in this chapter. This is illustrated by Table 5.4, where we show for several grounders the impact of manually adding redundant information. For all the grounders in this table except GIDL, manually adding redundancy may have a serious impact. For some grounders, the need to add redundancy can sometimes be avoided by writing the input theory in a specific format. For example, the grounder GRINGO (Gebser et al., 2007) uses a syntactic check to derive bounds: it derives that predicate `q` of the input vocabulary is a bound for predicate `p` if `p` is defined by a choice rule of the form, e.g., `{p(X)} :- q(X)`. However, if this rule is replaced by `{p(X)} :- dom(X)`, where `dom` denotes the domain, and the constraint `:- p(X),not q(X),dom(X)` is added, `q` is still a bound for `p`, but this is not detected by GRINGO, as can be seen in Table 5.4.
The grounder of the DLV system (Perri et al., 2007) may derive bounds by reasoning on the propositional level. As we explain below, the order in which rules and constraints are grounded is of crucial importance for such a method to pay off. Since DLV grounds rules before constraints, using a constraint to state that `q` is a bound for `p` does not improve grounding time.

Propagation on the Propositional Level One of the techniques to optimize grounding consists of applying a constraint propagation method on the ground theory T_g and replacing by $\top$, respectively $\bot$, every ground literal that is derived to be true, respectively false. The resulting theory is then simplified. This technique is applied by the grounder PSGRND (East et al., 2006), which uses *unit propagation* (Davis and Putnam, 1960) and complete *one-atom lookahead* (Li and Anbulagan, 1997) as propagation methods. The latter is performed

[26] `www.logicpalet.com`

	constr	redun	defin
GRINGO	76.33	1.59	0.60
DLV	339.37	4.23	2.81
LPARSE	63.25	0.78	63.58
PSGRND	44.79	0.72	n/a
GIDL	0.26	0.42	n/a

Table 5.4: Grounding times (in seconds) for the Hamiltonian circuit problem with an input graph of 200 nodes and 1800 edges. Encoding *constr* uses a constraint to state that each edge in the cycle should be an edge of the graph. Encoding *redun* adds redundancy to include this bound in all rules and constraints. Encoding *defin* contains no redundancy, but limits the possible edges in the cycle to the edges in the graph while defining the search space for the cycle.

once the grounding is finished, the former is triggered each time a unit clause is added to the grounding. If an inconsistency is detected by unit propagation, the grounding process is terminated immediately. Observe that this technique yields small groundings but does not improve grounding speed, except for the (rare) case where the propagation method detects an inconsistency during grounding. Indeed, it does not avoid computing all ground instances of the formulas in the input theory.
If a propositional constraint propagation method is applied while the grounding is being constructed, the derived information could be used to refine bounds. For instance, if unit-propagation derives that the domain atom $P(d_1, \ldots, d_n)$ is true, then $x_1 = d_1 \wedge \ldots \wedge x_n = d_n$ is a ct-bound for $P(x_1, \ldots, x_n)$. These bounds could be used to speed up the construction of the rest of the grounding. For this method to be effective, however, some careful fine-tuning of the order in which sentences are grounded is required. It may even be necessary to alternatingly compute partial groundings of different sentences. To the best of our knowledge, this process has not been worked out or implemented with unit-propagation or one-atom lookahead as underlying propagation method. On the other hand, most ASP grounders apply it for the following limited propagation method: if all rules defining a predicate P are grounded, it is concluded that a domain atom $P(\overline{d})$ is certainly true if it occurs in a ground rule of the form $P(\overline{d}) \leftarrow \top$, and certainly false if it does not occur in the head of any ground rule. In this case, a good grounding order can be derived from the dependency graph of the input theory (Cadoli and Schaerf, 2005; Perri et al., 2007). In GIDL, this strategy is implemented for grounding definitions.

Sharing A second technique is called *sharing* and consists of detecting subformulas in the ground theory T_g that occur more than once. If such a subformula φ is detected, all its occurrences in T_g are replaced by a new atom P, and the sentence $P \Leftrightarrow \varphi$ is added. If φ is a large formula and occurs often in T_g, this may result in a significant grounding size reduction. Also, sharing improves the

propagation in SAT solvers.
Shlyakhter et al. (2003b) present an algorithm to detect identical subformulas on the first-order level, Torlak and Jackson (2007) for the propositional level. In GIDL, we implemented a simple sharing technique using dynamic programming. We adapted function `groundConj` so that instead of returning a conjunction $\bigwedge C$, it creates a new atom P, adds the sentence $P \Leftrightarrow \bigwedge C$ to the grounding, and returns P. If `groundConj` is applied multiple times on the same input φ, the same predicate P is returned each time, but $P \Leftrightarrow \bigwedge C$ is added only once. Function `groundDisj` is adapted in similar fashion.

Clause splitting Clause splitting is a well-known rewriting technique applied in MACE style model generation (McCune, 2003). It consists of splitting a first-order clause

$$\forall x \forall y \forall \overline{z}\ (\varphi_1[x, \overline{z}_1] \vee \varphi_2[y, \overline{z}_2]) \tag{5.11}$$

where $x \notin \overline{z}_2$, $y \notin \overline{z}_1$ and $\overline{z} = \overline{z}_1 \cup \overline{z}_2$ into two new clauses

$$\forall x \forall \overline{z}_1\ (\varphi_1[x, \overline{z}_1] \vee S(\overline{z}_1 \cap \overline{z}_2)) \tag{5.12}$$

$$\forall y \forall \overline{z}_2\ (\neg S(\overline{z}_1 \cap \overline{z}_2) \vee \varphi_2[y, \overline{z}_2]). \tag{5.13}$$

Here, S is a new predicate symbol. The full grounding of (5.11) is of the size $\mathcal{O}(|D|^3)$, while the full grounding of (5.12) and (5.13) has only size $\mathcal{O}(|D|^2)$.
If sharing is implemented by adapting `groundConj` and `groundDisj` as explained above, the effect of clause splitting can be obtained by moving quantifiers according to the equivalences (2.3), (2.4), (2.7) and (2.8) of Section 2.1.3. For instance, we can apply equivalences (2.3) and (2.7) to replace (5.11) by $\forall x \forall \overline{z}\ (\varphi_1 \vee (\forall y\ \varphi_2))$. Grounding the latter while applying sharing has the same effect as clause splitting. Similarly, the grounding size of $\exists x \exists y \exists \overline{z}\ (\varphi_1[x, \overline{z}_1] \wedge \varphi_2[y, \overline{z}_2])$ can be reduced by replacing this formula by $\exists x \exists \overline{z}\ (\varphi_1 \wedge (\exists y\ \varphi_2))$.
The simple heuristic to guide clause splitting described by Claessen and Sörensson (2003) can directly be applied to choose which quantifiers to move inside. We conclude that clause splitting could easily be incorporated in GIDL.

Database Techniques Several query optimization techniques in databases can be used to optimize grounding. Examples are join-ordering strategies, backjumping and indexing techniques.
One of the most basic techniques to improve grounding speed consists of reordering (long) conjunctions or disjunctions of literals to speed up grounding. Which order is best depends on the grounding algorithm. Different strategies are described by, e.g, Leone et al. (2001); Syrjänen (1998, 2009) and in the database literature (Garcia-Molina et al., 2000). There is no problem implementing a similar technique in GIDL. Also, reordering the nodes in the BDD representation of the bounds could optimize querying. It is part of future work to investigate such reordering strategies for BDDs.
One of the important methods in the DLV grounder is the use of a backjumping technique (Perri et al., 2007) to efficiently find all instances of a conjunction

$\varphi_1 \wedge \ldots \wedge \varphi_n$ that are possibly true, given (an overestimation of) the possibly true instances of each of the conjuncts φ_i. In GIDL, this backjumping technique is applied to implement line 8 of function `groundDisj` in the case where φ is a conjunction. Indeed, if φ is the formula $\varphi_1 \wedge \ldots \wedge \varphi_n$, then line 8 amounts to finding all possibly true instances of φ, while the cf-bounds for φ_1, ..., φ_n provide an overestimation of the possibly true instances of these conjuncts. Similarly, the backjumping technique is applied to improve line 8 of `groundConj` in case φ is a disjunction.
Catalano et al. (2008) present an adaptation of indexing strategies for grounding.

Partition-Based Reasoning Ramachandran and Amir (2005) describe a sophisticated grounding technique that can reduce the grounding size of FO theories, depending on the availability of some graphical structure in these theories. This technique is not directly applicable in our case, since it produces groundings that are not necessarily I_σ-equivalent to the input theory. The only guarantee is that the ground theory is satisfiable iff the input problem is satisfiable.

Calculating known definitions We say that a definition Δ *depends on expansion symbols* if $\mathrm{Open}(\Delta) \not\subseteq \sigma$. If Δ does not depend on expansion symbols, then the interpretation of every predicate in $\mathrm{Def}(\Delta)$ is the same in every model M of T expanding I_σ. Indeed, for such a definition and any $M \models_{I_\sigma} T$, $M|_{\mathrm{Open}(\Delta)}$ is completely determined by I_σ. Therefore also $\mathrm{wfm}_\Delta(M)$ only depends on I_σ. This fact can be exploited to optimize grounding. Assume Δ is a definition that does not depend on expansion symbols. Let τ be the vocabulary $\langle \sigma_{\mathrm{sort}}, \sigma_{\mathrm{pred}} \cup \mathrm{Def}(\Delta), \sigma_{\mathrm{func}} \rangle$ and I_τ the unique τ-structure such that $I_\tau|_\sigma = I_\sigma$ and $I_\tau \models \Delta$. Then clearly, $M \models_{I_\sigma} T$ iff $M \models_{I_\tau} T$ for any structure M. However, a grounding for $T \setminus \Delta$ with respect to τ can be computed more efficiently, since $\mathrm{Gr}_{\mathrm{red}}(T \setminus \Delta, I_\tau)$ is necessarily smaller than $\mathrm{Gr}_{\mathrm{red}}(T, I_\sigma)$. Indeed, $T \setminus \Delta$ is a subtheory of T, and $\mathrm{Gr}_{\mathrm{red}}(T \setminus \Delta, I_\tau)$ does not contain symbols of $\mathrm{Def}(\Delta)$, while $\mathrm{Gr}_{\mathrm{red}}(T, I_\sigma)$ does. Current grounders iteratively remove in this way all definitions that do not depend on expansion symbols before the grounding phase.
The deductive database literature describes several algorithms to compute the well-founded model $\mathrm{wfm}_\Delta(M)$ for a definition that does not depend on expansion symbols. Most of them are only defined for definitions where every rule body is a conjunction of atoms, but some of them, such as the the semi-naive evaluation technique (Ullman, 1988), can easily be adapted to handle full FO bodies.[27] In GIDL, we implemented a simple adaptation of the semi-naive technique. Currently, the implementation is far from optimized, which resulted in a low ranking of GIDL on the category *Decision problems in* **P** of the latest ASP competition (Denecker et al., 2009). It is part of future work to apply (at least in some cases) XSB (Swift, 2009) instead of our implementation of the semi-naive technique in GIDL.

[27] See also the discussion below Proposition 4.10

In the case a c-map is computed, calculating the interpretations of definitions that do not depend on expansion symbols is beneficial for another reason. Let τ be the vocabulary introduced above. Then the set of c-maps for T over τ is a superset of the set of c-maps for T over σ, since the bounds assigned by the former c-maps are formulas over τ, instead of only over σ. As such, c-maps computed by symbolic propagation for T over τ might yield more efficient grounding compared to c-maps computed for T over σ.

5.5.2 Grounders

A non-native approach to ground an MX(FO(ID)) problem consists of first translating it to an equivalent normal logic program under the well-founded semantics. This translation is described by Mariën et al. (2004). Next, a (slightly adapted) grounder for ASP is used to ground the logic program. This is the approach taken by MXIDL (Mariën et al., 2006).

The first native grounding algorithm for MX(FO) and MX(FO(ID)) was described by Patterson et al. (2007). It is based on relational algebra and takes a "bottom-up approach" (see Section 5.1). To construct a grounding of a sentence φ, it first creates all possible groundings of the atomic subformulas. Then it combines these groundings using relational algebra operations, working its way up the syntax tree. Finally, a grounding for φ is obtained. Mitchell et al. (2006) report on an implementation, called MXG, of the algorithm.

KODKOD (Torlak and Jackson, 2007) is an MX grounder for a syntactic variant of FO. Like MXG, it works in a bottom-up way. It represents intermediate groundings by (sparse) matrices. One of the features of KODKOD is that it allows a user to give part of a solution to an MX problem as a three-valued structure. Specifically, the user can force that some domain atoms $P(\overline{d})$, where P is a predicate in the expansion vocabulary, are certainly true (or certainly false). KODKOD then takes advantage of this information to produce smaller groundings. GIDL also allows for a three-valued structure as input. When computing the c-map, the set V of tuples $\overline{d}$ for which the user indicates that P should be true is then used as initial ct-bound for P instead of $\bot$, i.e. the initial structure $\tilde{\Phi}_\sigma$ assigns $\{\overline{x} \mid V(\overline{x})\}$ as query to P^{ct}, instead of $\{\overline{x} \mid \bot\}$. Similarly for the cf-bound. This leads to more efficient and compact groundings.

MACE (McCune, 2003) and PARADOX (Claessen and Sörensson, 2003) are finite model generators for FO. They work by choosing a domain and grounding the input theory to SAT. If the resulting grounding is unsatisfiable, the domain size is increased and the process is repeated. The grounding algorithm in MACE and PARADOX basically constructs the full grounding and simplifies it afterwards. Small groundings are obtained by first rewriting the input theory using, e.g., clause splitting. Also methods that build the grounding *incrementally* are applied in these systems to avoid recomputing every grounding from scratch.

East et al. (2006) developed the grounder PSGRND for MX(PS^{pb}). PS^{pb} is a fragment of FO(ID), extended with pseudo-boolean constraints. As explained above, PSGRND performs reasoning on the ground theory to reduce memory usage and grounding size. The experiments performed by East et al. (2006)

show that carefully designed data structures are of key importance to build an efficient grounder.
ASP grounders take as input a normal logic program and transform it into an equivalent ground normal logic program. As such, these grounders do not deal with (deeply) nested formulas. Currently, there are three ASP grounders: LPARSE (Syrjänen, 2000; Syrjänen, 2009), GRINGO (Gebser et al., 2007) and the grounding component of DLV (Perri et al., 2007). All of them use techniques from database theory to perform grounding efficiently.
Finally, we mention the grounder SPEC2SAT (Cadoli and Schaerf, 2005). Its input theories are in the NP-SPEC language, a language with Datalog-like syntax and semantics based on model minimality.
It would be interesting to compare the efficiency of the mentioned grounders experimentally. However, it is currently not possible to conduct such an experiment in a scientifically fair way. There are several reasons for this. First, all grounders have a different input language, making it impossible to run them on the same input. Also, there are several output languages for grounders. A richer output language leads to more compact and fast grounding. For instance, for some problems, LPARSE's output size is necessarily cubic in the input domain size, while GIDL's output format allows for quadratic size. Thirdly, even if the input and output languages of all grounders were the same, an expert could easily manipulate experiments by carefully choosing his modelling style. For example, if he does not manually add bounds to the input theories, GIDL has an advantage. If bodies of rules are not ordered, DLV is more likely to produce good results. Etc. Finally, because of the large amount of data processed by grounders, carefully designed data structures and an optimized implementation of the core grounding algorithm is very important to achieve fast grounding (East et al., 2006). However, several of the above mentioned grounders are not yet optimized in that sense. As such, it is difficult to derive conclusions about grounding *algorithms* by experimentally comparing the efficiency of current *implementations* of these algorithms.

5.6 Conclusion

In this chapter, we investigated how to improve the grounding (or propositionalization) of a logic theory by automatically adding redundant information to the theory. We showed that this redundant information can be derived by the symbolic propagation method we presented in Chapter 4 and we obtained results about where the information can be added to a theory. We investigated properties of redundant information that ensure smaller groundings.
As a case study, we implemented our algorithm for computing and adding redundant information on top of a simple top-down style grounding algorithm. Experiments showed that if the redundant information is carefully restricted, not only smaller groundings are obtained, but that the groundings are often computed much faster.

Chapter 6

Debugging

One of the benefits of using finite model generation to solve computational problems is that typically, the logic theory describing the problem is more compact and readable than a program to solve the same problem in a standard programming language. Nevertheless, bugs are made when writing theories and debugging is often difficult in practice, since debugging methods for logic theories are still in their infancy.
Bugs in theories manifest themselves in two different ways, which require a different sort of debugging support. A bug causes a model generator to either produce an unintended model, or to omit an intended one. To illustrate the former type of bug, consider the following theory, expressing the constraints on a proper graph colouring.

$$\forall v \exists c \; Col(v, c). \tag{6.1}$$

$$\forall v \forall c_1 \forall c_2 \; (Col(v, c_1) \wedge Col(v, c_2) \Rightarrow c_1 = c_2). \tag{6.2}$$

$$\forall v_1 \forall v_2 \forall c_1 \forall c_2 \; (Col(v_1, c_1) \wedge Col(v_2, c_2) \wedge Edge(v_1, v_2) \Rightarrow c_1 \neq c_2) \tag{6.3}$$

Here, $Col(v, c)$ is used to denote that vertex v has colour c and $Edge(v_1, v_2)$ means that there is an edge between the vertices v_1 and v_2. Assume that a user makes a typo in sentence (6.2) and writes the tautology

$$\forall v \forall c_1 \forall c_2 \; (Col(v, c_1) \wedge Col(v, c_2) \Rightarrow \underline{c_1 = c_1}) \tag{6.4}$$

instead. This will cause a model generator to produce models where some nodes have more than one colour. By inspecting these models, a user can deduce that the bug is located in sentence (6.4), since this is the constraint that should express that a vertex has at most one colour. The second type of bug is often more difficult to locate. For example, if a user makes the typical mistake of assuming that variables with different names take different values (Shlyakhter et al., 2003a), and therefore writes

$$\forall v \forall c_1 \forall c_2 \; \neg(Col(v, c_1) \wedge Col(v, c_2)). \tag{6.5}$$

instead of (6.2), then a model generator will answer that the problem has no solution. Indeed, (6.5) forces that $Col(v, c)$ is false for every v and c, which contradicts (6.1). Observe that a user has no clue where to search for a bug now. In this chapter, we mainly focus on debugging support for the second type of bugs. We briefly discuss systems that facilitate locating the first type of bugs by visualizing finite structures.

The most used approach to debug programs in a standard programming language is by analyzing the *trace*, i.e., the sequence of steps performed while running the program. Also, debugging by analyzing a trace has proven to be useful in many declarative programming contexts such as Prolog (Shapiro, 1983; Ducassé, 1999), Haskell (Nilsson, 1999), ILP (Tronçon and Janssens, 2007), constraint programming (Meier, 1995) and deductive databases (Mallet and Ducassé, 1999). This suggests to debug logic theories by analyzing the trace of a model generator. However, to make such an approach work, the model generator should satisfy the following two requirements:

1. All reasoning steps should be as simple as possible. At least, they should be clear for someone who knows the informal semantics of the used logic.

2. All reasoning should be shown on the original theory as provided by the user. If the model generator relies on a (preprocessing) phase where the theory is brought into some normal form, it should be possible to translate its reasoning back into reasoning on the original theory. Indeed, reasoning on a transformed theory is not transparent for a user.

In this chapter, we show that the model generation algorithm presented in Chapter 4 (Algorithm 4.4) satisfies these two requirements. Hence, it allows for debugging by analyzing the trace. To this end, we present a formal proof system for model generation, which is called the *MX-calculus* and is introduced in Section 6.1. If a model generation problem has no solutions, the trace of the model generator corresponds to an MX-calculus proof for the inconsistency. If a model is found, this model can easily be extracted from the trace. In Section 6.2, we present two techniques to further facilitate debugging. The first one allows a user to describe (part of) expected models that were omitted by a model generator. This yields smaller, and hence more comprehensive, proofs. The other technique consists of an interactive sessions that guides the user to relevant parts of a proof. To show that our debugging approach can be used for richer logics than FO, we extend in Section 6.3 the MX-calculus to FO(ID). Finally, we compare our debugging approach with the (very different) approaches proposed for ASP and the Alloy language.

6.1 The finite model generation calculus

As mentioned above, the debugging method we propose relies on the ability of a model generator to output a *proof* of inconsistency in case its input is an unsatisfiable problem. In this section, we present a formal proof system, called

the *MX-calculus* to represent such a proof. The proof system is inspired by tableau calculi (D'Agostino et al., 1999).
In the rest of this chapter, we use the three-valued version of model expansion (Definition 4.16). We recall its definition.

Definition 6.1. Let T be a theory over Σ and $\tilde{I}$ a finite three-valued Σ-structure. The model expansion search problem with input $\langle T, \tilde{I}\rangle$ is the problem of computing a two-valued Σ-structure M such that $\tilde{I} \leq_p M$ and $M \models T$.

We denote by $M \models_{\tilde{I}} T$ that M is a model of T that is more precise than $\tilde{I}$, i.e., $M \models_{\tilde{I}} T$ means that M is a solution to the model expansion problem with input $\langle T, \tilde{I}\rangle$. We say that $\langle T, \tilde{I}\rangle$ is *inconsistent* if there is no solution to the model expansion problem with input $\langle T, \tilde{I}\rangle$.

6.1.1 MX-trees

For the rest of this section, fix an FO theory T and a finite three-valued structure $\tilde{I}$. We consider the MX problem with input $\langle T, \tilde{I}\rangle$. Proofs for $\langle T, \tilde{I}\rangle$ in the MX-calculus are built using rules of the form

$$\frac{\mathcal{I}_1, \ldots, \mathcal{I}_n}{\mathcal{J}_1 \mid \ldots \mid \mathcal{J}_m} \tag{6.6}$$

where $\mathcal{I}_1, \ldots, \mathcal{I}_n, \mathcal{J}_1, \ldots, \mathcal{J}_m$ are *signed instances*: pairs of an instance $\varphi[\overline{x}/\overline{d}]$ and a positive ($\oplus$) or negative ($\ominus$) *sign*. Signed instances are denoted by $\varphi[\overline{x}/\overline{d}]^{\oplus}$ or $\varphi[\overline{x}/\overline{d}]^{\ominus}$. We call $\mathcal{I}_1, \ldots, \mathcal{I}_n$ the *premises* of the rule, and $\mathcal{J}_1, \ldots, \mathcal{J}_m$ its *consequences*. Intuitively, the rule means that if all its positive, respectively negative, premises are true, respectively false, then at least one of its consequences is positive and true or negative and false. A rule is *sound* if its intuitive meaning is indeed a sound reasoning. More precisely:

Definition 6.2. If $\mathcal{I}$ is the signed instance $\varphi[\overline{x}/\overline{d}]^{\oplus}$, respectively $\varphi[\overline{x}/\overline{d}]^{\ominus}$, then denote by $\mathfrak{S}(\mathcal{I})$ the instance $\varphi[\overline{x}/\overline{d}]$, respectively $\neg\varphi[\overline{x}/\overline{d}]$. A rule of the form (6.6) is *sound with respect to T and $\tilde{I}$* if for every $M \models_{\tilde{I}} T$ such that M satisfies $\bigwedge_{1\leq i\leq n} \mathfrak{S}(\mathcal{I}_i)$, M also satisfies $\bigvee_{1\leq i\leq m} \mathfrak{S}(\mathcal{J}_i)$.

We distinguish between three types of rules in the MX-calculus: initialization, propagation and cut rules. All of them are sound with respect to $\langle T, \tilde{I}\rangle$.

Initialization rules

The following are the seven initialization rules for $\langle T, \tilde{I}\rangle$. None of them has premises.

$$\text{(I+↓)}\ \frac{}{\varphi^{\oplus}} \qquad \text{(I-↑)}\ \frac{}{d = d'^{\ominus}} \qquad \text{(I-↑)}\ \frac{}{Q(\overline{d})^{\ominus}} \qquad \text{(I-↑)}\ \frac{}{G(\overline{d}) = d^{\ominus}}$$

$$\text{(I+↑)}\ \frac{}{d = d^{\oplus}} \qquad \text{(I+↑)}\ \frac{}{P(\overline{d})^{\oplus}} \qquad \text{(I+↑)}\ \frac{}{F(\overline{d}) = d^{\oplus}}$$

Here, φ is a sentence of T or the sentence $\top$, d and d' are two different domain elements, $P(\overline{d})$ is a domain atom such that $P^{\tilde{I}}(\overline{d}) = \mathbf{t}$, $Q(\overline{d})$ is an atom such that $Q^{\tilde{I}}(\overline{d}) = \mathbf{f}$, F is a function such that $F^{\tilde{I}}(\overline{d}) = \{d\}$ and G is a function such that $d \notin G^{\tilde{I}}(\overline{d})$. Intuitively, rule (I+↓) expresses that a sentence of T is necessarily true. The other rules assert the truth value in $\tilde{I}$ of an atom that is not unknown in $\tilde{I}$.

Propagation rules

If the domain D of $\tilde{I}$ is given by $D = \{d_1, \ldots, d_n\}$, the following are all propagation rules for $\langle T, \tilde{I}\rangle$.

- Negation rules:

$$(\neg\text{-}\downarrow)\ \frac{\neg\varphi^{\oplus}}{\varphi^{\ominus}} \qquad (\neg\text{-}\uparrow)\ \frac{\varphi^{\oplus}}{\neg\varphi^{\ominus}} \qquad (\top+)\ \frac{\top^{\oplus}}{\bot^{\ominus}}$$

$$(\neg+\downarrow)\ \frac{\neg\varphi^{\ominus}}{\varphi^{\oplus}} \qquad (\neg+\uparrow)\ \frac{\varphi^{\ominus}}{\neg\varphi^{\oplus}} \qquad (\bot+)\ \frac{\bot^{\oplus}}{\top^{\ominus}}$$

- Conjunction rules (where $j \in [1, m]$):

$$(\wedge+\uparrow)\ \frac{{\varphi_1}^{\oplus}, \ldots, {\varphi_m}^{\oplus}}{{\textstyle\bigwedge_{i\in[1,m]}\varphi_i}^{\oplus}} \qquad (\wedge\text{-}\uparrow)\ \frac{{\varphi_j}^{\ominus}}{{\textstyle\bigwedge_{i\in[1,m]}\varphi_i}^{\ominus}} \qquad (\wedge+\downarrow)\ \frac{{\textstyle\bigwedge_{i\in[1,m]}\varphi_i}^{\oplus}}{{\varphi_j}^{\oplus}}$$

$$(\wedge\text{-}\downarrow)\ \frac{\left(\bigwedge_{i\in[1,m]}\varphi_i\right)^{\ominus}, {\varphi_1}^{\oplus}, \ldots, {\varphi_{j-1}}^{\oplus}, {\varphi_{j+1}}^{\oplus}, \ldots, {\varphi_j}^{\oplus}}{{\varphi_j}^{\ominus}}$$

- Disjunction rules (where $j \in [1, m]$):

$$(\vee\text{-}\uparrow)\ \frac{{\varphi_1}^{\ominus}, \ldots, {\varphi_m}^{\ominus}}{{\textstyle\bigvee_{i\in[1,m]}\varphi_i}^{\ominus}} \qquad (\vee+\uparrow)\ \frac{{\varphi_j}^{\oplus}}{{\textstyle\bigvee_{i\in[1,m]}\varphi_i}^{\oplus}} \qquad (\vee\text{-}\downarrow)\ \frac{{\textstyle\bigvee_{i\in[1,m]}\varphi_i}^{\ominus}}{{\varphi_j}^{\ominus}}$$

$$(\vee+\downarrow)\ \frac{\left(\bigvee_{i\in[1,m]}\varphi_i\right)^{\oplus}, {\varphi_1}^{\ominus}, \ldots, {\varphi_{j-1}}^{\ominus}, {\varphi_{j+1}}^{\ominus}, \ldots, {\varphi_j}^{\ominus}}{{\varphi_j}^{\oplus}}$$

- Universal rules:

$$(\forall+\uparrow)\ \frac{\varphi[x/d_1]^{\oplus}, \ldots, \varphi[x/d_n]^{\oplus}}{\forall x\ \varphi[x]^{\oplus}}$$

$$(\forall\text{-}\uparrow)\ \frac{\varphi[x/d_i]^{\ominus}}{\forall x\ \varphi[x]^{\ominus}} \qquad\qquad (\forall\text{+}\downarrow)\ \frac{\forall x\ \varphi[x]^{\oplus}}{\varphi[x/d_i]^{\oplus}}$$

$$(\forall\text{-}\downarrow)\ \frac{\forall x\ \varphi[x]^{\ominus}, \varphi[x/d_1]^{\oplus}, \ldots, \varphi[x/d_{i-1}]^{\oplus}, \varphi[x/d_{i+1}]^{\oplus}, \ldots, \varphi[x/d_n]^{\oplus}}{\varphi[x/d_i]^{\ominus}}$$

- Existential rules:

$$(\exists\text{-}\uparrow)\ \frac{\varphi[x/d_1]^{\ominus}, \ldots, \varphi[x/d_n]^{\ominus}}{\exists x\ \varphi[x]^{\ominus}}$$

$$(\exists\text{+}\uparrow)\ \frac{\varphi[x/d_i]^{\oplus}}{\exists x\ \varphi[x]^{\oplus}} \qquad\qquad (\exists\text{-}\downarrow)\ \frac{\exists x\ \varphi[x]^{\ominus}}{\varphi[x/d_i]^{\ominus}}$$

$$(\exists\text{+}\downarrow)\ \frac{\exists x\ \varphi[x]^{\oplus}, \varphi[x/d_1]^{\ominus}, \ldots, \varphi[x/d_{i-1}]^{\ominus}, \varphi[x/d_{i+1}]^{\ominus}, \ldots, \varphi[x/d_n]^{\ominus}}{\varphi[x/d_i]^{\oplus}}$$

- Equality rules:

$$(=\pm\downarrow)\ \frac{\mathcal{I}, t_1 = t_2{}^{\oplus}}{\mathcal{I}'}$$

  where $\mathcal{I}'$ is the result of replacing in $\mathcal{I}$ an occurrence of t_1 by t_2, or an occurrence of t_2 by t_1.

- Function rules (where $d_j \neq d_k$):

$$(\text{F-}\downarrow)\ \frac{F(\overline{d}) = d_j{}^{\oplus}}{F(\overline{d}) = d_k{}^{\ominus}}$$

$$(\text{F+}\downarrow)\ \frac{F(\overline{d}) = d_1{}^{\ominus}, \ldots, F(\overline{d}) = d_{i-1}{}^{\ominus}, F(\overline{d}) = d_{i+1}{}^{\ominus}, \ldots, F(\overline{d}) = d_n{}^{\ominus}}{F(\overline{d}) = d_i{}^{\oplus}}$$

We stress that each of these rules is easy to understand. For instance, disjunction rule ($\vee$+$\uparrow$) says that a disjunction is true if one of its disjuncts is true. Universal rule ($\forall$-$\downarrow$) says that if a formula $\forall x\ \varphi[x]$ is false, but for all domain elements d except d_i, the instance $\varphi[d]$ is true, then $\varphi[d_i]$ is false. Indeed, if $\varphi[d_i]$ would be true, then also $\forall x\ \varphi[x]$ would be true.

Cut rule

The cut rule for $\langle T, \tilde{I}\rangle$ is given by $\dfrac{}{\varphi[\overline{x}/\overline{d}]^{\oplus} \quad | \quad \varphi[\overline{x}/\overline{d}]^{\ominus}}$

Definition 6.3. An *MX-rule* is an initialization, propagation or cut rule.

Lemma 6.1. *Every MX-rule is sound.*

Proof. The proof is straightforward for each of the different MX-rules. ■

6.1.2 Soundness and completeness

An MX-calculus proof for the inconsistency of $\langle T, \tilde{I}\rangle$ is a tree, built using the rules defined above, such that each of its branches[28] contains a contradiction. Formally, it is defined as follows.

Definition 6.4. An *MX-tree for* $\langle T, \tilde{I}\rangle$ is inductively defined by

- the empty tree is an MX-tree for $\langle T, \tilde{I}\rangle$;
- if $\mathcal{T}$ is an MX-tree for $\langle T, \tilde{I}\rangle$, B a branch of $\mathcal{T}$ and $\dfrac{\mathcal{I}_1, \ldots, \mathcal{I}_n}{\mathcal{J}_1 \mid \ldots \mid \mathcal{J}_m}$ an MX-rule for $\langle T, \tilde{I}\rangle$ such that all $\mathcal{I}_i$ occur in B, then the result of adding in $\mathcal{T}$ all $\mathcal{J}_1 \ldots \mathcal{J}_m$ as children to the leaf of B is an MX-tree for $\langle T, \tilde{I}\rangle$.

Example 6.1. Let T_1 be the theory consisting of sentence (6.1), (6.3) and (6.5), and let $\tilde{I}_1$ be a three-valued structure with domain D_1 containing precisely the two colours **red** and **blue**, and at least one node **d**. Assume $Col^{\tilde{I}_1}(\mathbf{d}, \mathbf{blue}) = \mathbf{f}$. Figure 6.1 shows an MX-tree for $\langle T_1, \tilde{I}_1\rangle$. The used MX-rules and premises are indicated next to each node.

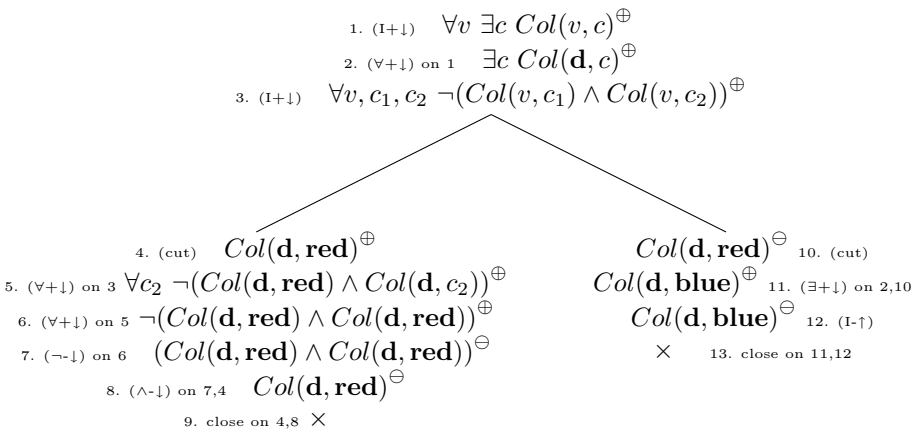
We say that a branch of an MX-tree is *closed* if for some instance $\varphi[\overline{x}/\overline{d}]$, it contains both $\varphi[\overline{x}/\overline{d}]^{\oplus}$ and $\varphi[\overline{x}/\overline{d}]^{\ominus}$. We call $\varphi[\overline{x}/\overline{d}]^{\oplus}$ and $\varphi[\overline{x}/\overline{d}]^{\ominus}$ *conflicting instances* of that branch. For example, the left branch of the tree in Figure 6.1 is closed because it contains the conflicting instances $Col(\mathbf{red}, \mathbf{d})^{\oplus}$ and $Col(\mathbf{red}, \mathbf{d})^{\ominus}$. We indicate a closed branch with the symbol $\times$. An MX-tree is closed if all its branches are closed. An *MX-proof for* $\langle T, \tilde{I}\rangle$ is a closed MX-tree for $\langle T, \tilde{I}\rangle$. The next theorem states the soundness and completeness of the MX-calculus.

Theorem 6.2. *There exists an MX-proof for* $\langle T, \tilde{I}\rangle$ *iff there exists no structure* M *such that* $M \models_{\tilde{I}} T$.

Proof. See Appendix A. ■

[28] In this text, a *branch* of a tree is a path from the root of the tree to one of the leaves.

Figure 6.1: An MX-tree for Example 6.1.



6.1.3 Saturated branches

Although Theorem 6.2 guarantees the existence of an MX-proof for $\langle T, \tilde{I}\rangle$ if there is no model of T approximated by $\tilde{I}$, a naive implementation of the MX-calculus could never find such a proof. This is even the case if the same formula is never added twice to a branch. An implementation could, e.g., endlessly apply the cut rule for different instances. However, as we will show below, we can restrict the instances that may occur in an MX-tree in such a way that termination is guaranteed. Moreover, if such a restricted MX-tree $\mathcal{T}$ is not closed and cannot be extended, a model of T can easily be extracted from $\mathcal{T}$.
For a theory T over Σ, we call an instance a *T-instance* if it is either an instance of a subformula of T, or an atom of the form $P(\overline{d})$, $F(\overline{d}) = d$ or $d = d$. A rule of the form (6.6) is *T-restricted* if all $\mathcal{J}_i$ are signed T-instances. We call a branch B of an MX-tree *T-saturated* if for every T-restricted MX-rule of the form (6.6) that can be applied to B, at least one of the $\mathcal{J}_i$ already occurs in B.

Definition 6.5. Let B be a non-closed branch of an MX-tree for $\langle T, \tilde{I}\rangle$. The *implicit structure of B* is the three-valued structure $\tilde{B}$ with the same domain as $\tilde{I}$ and defined by

$$P^{\tilde{B}}(\overline{d}) = \begin{cases} \mathbf{t} & \text{if } P(\overline{d})^{\oplus} \in B \\ \mathbf{f} & \text{if } P(\overline{d})^{\ominus} \in B \\ \mathbf{u} & \text{otherwise} \end{cases}$$

and $F^{\tilde{B}}(\overline{d}) = \{d' \mid F(\overline{d}) = d'^{\ominus} \notin B\}$ for every predicate P, function F and domain elements $\overline{d}, d'$.

Lemma 6.3. *Let B be a non-closed T-saturated branch, and $\tilde{B}$ its implicit structure. Then for any two-valued structure $M \geq_p \tilde{B}$, it holds that $M \models T$.*

Proof. See Appendix A ■

According to this lemma, Theorem 6.2 can be refined as follows.

Theorem 6.4. *There exists an MX-proof for $\langle T, \tilde{I}\rangle$ containing only signed T-instances iff there is no $M \geq_p \tilde{I}$ that satisfies T.*

Proof. The implication from left to right follows from theorem 6.2. For the other direction, let $\mathcal{T}$ be a saturated MX-tree containing only signed T-instances. Such a tree can be constructed by starting from the empty tree and applying T-restricted rules until saturation. This process yields a finite tree since there are only a finite number of T-instances. By Lemma 6.3, $\mathcal{T}$ is closed if no $M \geq_p \tilde{I}$ satisfies T. ■

The proof of Theorem 6.4 gives a simple procedure to find MX-proofs: start from the empty tree and keep applying T-restricted rules. This procedure terminates because there are only finitely many signed T-instances. If the resulting tree $\mathcal{T}$ is not closed, then there is no MX-proof for $\langle T, \tilde{I}\rangle$, and Lemma 6.3 provides an easy way to derive from $\mathcal{T}$ a model of T approximated by $\tilde{I}$.

6.1.4 Generating proofs by finite model generation

The finite model generation algorithm of Section 4.5.1 (Algorithm 4.4) using the propagation algorithm (Algorithm 4.2) can easily be adapted so that it generates an MX-tree while searching for a model of T. If no model is found for input $\langle T, \tilde{I}\rangle$, this tree is an MX-proof for the inconsistency of $\langle T, \tilde{I}\rangle$. The reason this adaptation is possible is that the MX propagation rules correspond precisely to INF propagators derived from ENF sentences, while the cut rule corresponds to the choice and exploration of different options in lines 5–8 of the model generation algorithm.

Recall that the propagation algorithm starts by rewriting its input theory T into ENF using Algorithm 4.1. In this process, auxiliary predicates Aux are introduced and subformulas $\varphi[\overline{x}]$ of the original[29] input theory T are replaced by $Aux(\overline{x})$. We denote the formula that is replaced when introducing Aux by φ_{Aux}. If ψ is a formula containing auxiliary predicates, we denote by $\lceil\psi\rceil$ the formula obtained from ψ by replacing each atom $Aux(\overline{x})$ over an auxiliary predicate by $\varphi_{Aux}[\overline{x}]$.

Now we introduce the adaptations to the model generation algorithm on input $\langle T, \tilde{I}\rangle$ to create a tree $\mathcal{T}$. At each moment during the construction of $\mathcal{T}$, one of its branches is called the *active branch*. The adaptations are as follows[30]:

1. Before the first propagation takes place, build a T-restricted MX-tree $\mathcal{T}$ by exhaustively applying all initialization rules. Note that $\mathcal{T}$ has only one branch, which is therefore the active branch.

[29] Here we assume that the first step of Algorithm 4.1 is not executed, i.e., negations are not moved inside. Recall that moving the negations is not necessary for the correctness of Algorithm 4.1.

[30] Pseudo-code for the resulting algorithm can be found in Appendix A, Algorithm A.1

2. If during propagation an atom $P(\overline{d})$ is made true then the signed instances $\lceil P(\overline{d})\rceil^{\oplus}$ and $\lceil \neg P(\overline{d})\rceil^{\ominus}$ are added at the end of the active branch. Similarly, if an atom $P(\overline{d})$ is made false then the signed instances $\lceil P(\overline{d})\rceil^{\ominus}$ and $\lceil \neg P(\overline{d})\rceil^{\oplus}$ are added. If $P(\overline{d})$ is made inconsistent, both $\lceil P(\overline{d})\rceil^{\oplus}$ and $\lceil P(\overline{d})\rceil^{\ominus}$ are added to the active branch.

3. If a choice $P(\overline{d})$ is made (line 5 of Algorithm 4.4), then the cut rule is applied on the active branch to add the signed instances $\lceil P(\overline{d})\rceil^{\oplus}$ and $\lceil P(\overline{d})\rceil^{\ominus}$. The branch containing $\lceil P(\overline{d})\rceil^{\oplus}$ becomes the active branch. If line 8 is reached, the branch containing $\lceil P(\overline{d})\rceil^{\ominus}$ becomes the active branch.

Observe that the constructed tree keeps track of all the steps taken by the algorithm in its search for a model. In other words, the constructed tree for $\langle T, \tilde{I}\rangle$ represents a trace of executing the algorithm on $\langle T, \tilde{I}\rangle$.

Proposition 6.5. *If Algorithm 4.4 is adapted to generate an tree $\mathcal{T}$ as described above, then the generated tree for input $\langle T, \tilde{I}\rangle$ is an MX-tree for $\langle T, \tilde{I}\rangle$. Moreover, if Algorithm 4.4 returns a strictly four-valued structure, then $\mathcal{T}$ is an MX-proof for $\langle T, \tilde{I}\rangle$.*

Proof. See Appendix A ■

We made a prototype implementation of the above algorithm to construct MX-proofs. In our implementation, propositional binary decision diagrams are used to represent large sets of instances in a compact way. Since no heuristics were implemented to guide the search, the implementation turns out to be a lot slower than IDP. On problems with bugs however, it is efficient enough to be useful in practice. Another approach to construct MX-proofs consists of translating the trace of a grounding based finite model generator such as IDP into an MX-proof. This approach is sketched in Appendix B and yields a system that is as efficient as the used finite model generator.

6.2 Debugging

As mentioned in the introduction, we propose a debugging system for locating and explaining bugs that cause a model generator to omit a number of expected models. In the extreme case, there is no model left because of the bug. Basically, the approach to locate and explain a bug in an input theory T and structure $\tilde{I}$ is as follows:

1. The user specifies a structure $\tilde{J} \geq_p \tilde{I}$ describing a class of expected models and such that there is no model of T approximated by $\tilde{J}$.

2. Construct a T-restricted MX-proof for $\langle T, \tilde{J}\rangle$.

3. Explore the proof to find the reason for $\langle T, \tilde{J}\rangle$ being inconsistent.

The first step serves three different purposes. From a technical point of view, it ensures that a proof can be constructed in the second step, since $\langle T, \tilde{J}\rangle$ is unsatisfiable. If $\langle T, \tilde{I}\rangle$ itself is unsatisfiable, one could take $\tilde{J}$ equal to $\tilde{I}$. From a user point of view, $\tilde{J}$ can describe a specific class of models that is missing. For example, in the graph colouring problem, a user can specify that he expects a solution where node **d** is red by assigning $Col^{\tilde{J}}(\mathbf{d}, \mathbf{red}) = \mathbf{t}$. Observe that if the user supplies a structure $\tilde{J}$ such that there exists an M such that $M \models_{\tilde{J}} T$, then instead of generating a proof, such a model M can be returned to indicate that there exists a solution that is in the class of structures described by $\tilde{J}$. Finally, the more precise $\tilde{J}$ is, the more concise proofs can be constructed for $\langle T, \tilde{J}\rangle$. Evidently, more concise proofs are more comprehensive.

We now elaborate on the third step of the debugging method, which is based on earlier work by Shapiro (1983). Since the proofs can be quite large, even for simple problems, it is necessary that only parts that are relevant for the user are shown. In particular, if a user understands that a certain instance $\varphi[\overline{x}/\overline{d}]$ is necessarily true in all solutions to the MX problem with input $\langle T, \tilde{J}\rangle$, then it is not needed to show him how the truth $\varphi[\overline{x}/\overline{d}]$ was actually derived. Also, all shown proof steps must be stated in natural language, so that they are comprehensible for a user who is not familiar with the MX-calculus.

The method we propose is an interactive session where the system provides a set S of signed instances that were derived (i.e., occur in the proof) to the user. Initially, the user is told that his input yields a conflict, and $S = \{\varphi^{\oplus}, \varphi^{\ominus}\}$, where $\varphi^{\oplus}$ and $\varphi^{\ominus}$ are conflicting instances of one of the branches of the proof. The user can then ask for an instance $\mathcal{I} \in S$ the reason why it was derived. Depending on $\mathcal{I}$, there are three possibilities:

- If a propagation rule was used to add $\mathcal{I}$ to the proof, the system's reply consists of (in natural language) that rule and its premises. The premises are added to the set S.
- If an initialization rule was used, the system replies that $\mathcal{I}$ was provided as input.
- If the cut rule was used and $\mathcal{I}$ is of the form $\varphi[\overline{x}/\overline{d}]^{\oplus}$, the system replies that there is a conflict if $\varphi[\overline{x}/\overline{d}]$ is assumed to be false. Conflicting instances of a branch below $\varphi[\overline{x}/\overline{d}]^{\ominus}$ are added to the set S. For instances $\mathcal{J}$ in S whose derivation depends on $\varphi[\overline{x}/\overline{d}]^{\ominus}$, the system indicates that $\mathcal{J}$ is derived under the assumption that $\varphi[\overline{x}/\overline{d}]^{\ominus}$ is false. Similarly if $\mathcal{I}$ is of the form $\varphi[\overline{x}/\overline{d}]^{\ominus}$.

We illustrate the method on the graph colouring example.

Example 6.2 (Example 6.1 ctd.)**.** Assume we want to detect the bug in theory T_1 and therefore explore the proof shown in Figure 6.1 for $\langle T_1, \tilde{I}_1\rangle$. Figure 6.2 shows a possible exploration of this proof. By the time the user asks question (3), he has most probably detected the bug. Since the derivation of (11) only

(**S** is the debugging system, **U** the user, the numbers refer to the nodes in the proof)

S: Your input implies that $Col(\mathbf{d}, \mathbf{red})$ must be true (4) and false (8) at the same time. $S = \{(4), (8)\}$.

U: (8) (i.e., the question: "Why must $Col(\mathbf{d}, \mathbf{red})$ be false?")

S: Because $Col(\mathbf{d}, \mathbf{red})$ must be true and $Col(\mathbf{d}, \mathbf{red}) \wedge Col(\mathbf{d}, \mathbf{red})$ must be false (7). $S = \{(4), (7)\}$.

U: (7)

S: Because $\neg(Col(\mathbf{d}, \mathbf{red}) \wedge Col(\mathbf{d}, \mathbf{red}))$ must be true (6). $S = \{(4), (6)\}$.

U: (6)

S: Because $\forall c_2 \neg(Col(\mathbf{d}, \mathbf{red}) \wedge Col(\mathbf{d}, c_2))$ must be true, and we can take $c_2 =$ **red**.

⋮

U: (3)

S: Because this is a sentence of the input theory. $S = \{(4)\}$.

U: (4)

S: If $Col(\mathbf{d}, \mathbf{red})$ is assumed to be false, then there is a conflict: $Col(\mathbf{d}, \mathbf{blue})$ must be true (11) and false (12). $S = \{(11)$ if (10) is assumed, $(12)\}$.

⋮

Figure 6.2: Interactively exploring a proof

holds when (10) is assumed, this is explicitly stated. Observe that (12), although it occurs below (10), does not use (10) in its derivation.

Choices and proof size

If the cut rule is often applied to construct a proof, or a proof is very large, it can be the case that — even with the exploration method outlined above — a user loses the overview. However, for several reasons we expect that this can often be avoided in practice:

- The *small scope hypothesis* (Jackson, 2006) claims that a bug typically occurs already in the context of structures $\tilde{I}$ with a small domain. For such a structure, a proof for $\langle T, \tilde{I} \rangle$ will be small too.

- A good algorithm to build the proofs should minimize the use of the cut rule by only applying it when no other rule can be applied. The algorithm sketched in Section 6.1.4 satisfies this requirement.
- If the description $\tilde{J}$ of the expected models is sufficiently precise, it is not needed to often apply the cut rule. In the extreme case where $\tilde{J}$ is two-valued, using the cut rule can be avoided altogether.

Nevertheless, in a worst case scenario, a bug is due to a combination of partially correct formulas and only shows up in large instances. Such bugs may be very hard to find and correct. This problem is inherent to debugging in all declarative languages.

6.3 Inductive definitions

In Section 6.1.4 we obtained the result that MX-trees correspond to a trace of the finite model generation algorithm of Chapter 4 on an FO theory. This indicates how we can extend the MX-calculus, and hence our debugging method to full FO($\cdot$): it suffices to add enough propagation rules to the calculus so that the trace of the model generation on an FO($\cdot$) theory corresponds to an MX-tree for this extended calculus. Below, we illustrate this by sketching how to extend the calculus to FO(ID).

6.3.1 Extending the MX-calculus to FO(ID)

To handle definitions in the MX-calculus, we extend it with three new classes of rules: *completion*, *unfounded set* and *non-totality* rules. As the names suggest, the completion rules handle the propagation that can be done on the completion of a definition, the unfounded set rules handle propagation due to unfounded sets and the non-totality rule makes sure a branch can be closed if the theory contains a definition that is non-total with respect to the implicit structure of that branch.

Completion rules

If Δ is a definition in the FO(ID) theory T, P a defined predicate of Δ and $\forall \overline{x}\ P(\overline{x}) \leftarrow \varphi_1[\overline{x}], \ldots, \forall \overline{x}\ P(\overline{x}) \leftarrow \varphi_n[\overline{x}]$ all rules in Δ with head P, then the following are the four completion rules for $\langle T, \tilde{I} \rangle$.

$$\frac{\varphi_1[\overline{x}/\overline{d}]^{\ominus}, \ldots, \varphi_n[\overline{x}/\overline{d}]^{\ominus}}{P(\overline{d})^{\ominus}}$$

$$\frac{\varphi_i[\overline{x}/\overline{d}]^{\oplus}}{P(\overline{d})^{\oplus}} \qquad\qquad \frac{P(\overline{d})^{\ominus}}{\varphi_i[\overline{x}/\overline{d}]^{\ominus}}$$

$$\frac{P(\overline{d})^{\oplus}, \varphi_1[\overline{x}/\overline{d}]^{\ominus}, \ldots, \varphi_{i-1}[\overline{x}/\overline{d}]^{\ominus}, \varphi_{i+1}[\overline{x}/\overline{d}]^{\ominus}, \ldots, \varphi_n[\overline{x}/\overline{d}]^{\ominus}}{\varphi_i[\overline{x}/\overline{d}]^{\oplus}}$$

Unfounded set rule

Let $\tilde{K}$ be a three-valued structure with the same domain as $\tilde{I}$, $\{Q_1(\overline{d}_1), \ldots, Q_n(\overline{d}_n)\}$ the set of all domain atoms that are true in $\tilde{K}$ and $\{Q_{n+1}(\overline{d}_{n+1}), \ldots, Q_m(\overline{d}_m)\}$ the set of all domain atoms that are false in $\tilde{K}$. Let Δ be a definition of T and U a set of domain atoms, defined with respect to Δ and unknown in $\tilde{K}$. If for $R(\overline{d}) \in U$ and any rule $\forall \overline{x}\ R(\overline{x}) \leftarrow \varphi[\overline{x}]$ in Δ, $\tilde{K}[U/\mathbf{f}](\varphi[\overline{x}/\overline{d}]) = \mathbf{f}$, then

$$\frac{Q_1(\overline{d}_1)^{\oplus}, \ldots, Q_n(\overline{d}_n)^{\oplus}, Q_{n+1}(\overline{d}_{n+1})^{\ominus}, \ldots, Q_m(\overline{d}_m)^{\ominus}}{P(\overline{d})^{\ominus}}$$

where $P(\overline{d}) \in U$, is an *unfounded set rule* for $\langle T, \tilde{I}\rangle$.

Non-totality rule

Let Δ be a definition of T and K be a two-valued Open(Δ)-structure with the same domain as $\tilde{I}$. Let $\{Q_1(\overline{d}_1), \ldots, Q_n(\overline{d}_n)\}$ the set of all domain atoms that are true in K and $\{Q_{n+1}(\overline{d}_{n+1}), \ldots, Q_m(\overline{d}_m)\}$ the set of all domain atoms that are false in K. If $\mathrm{wfm}_\Delta(K)$ is strictly three-valued, then

$$\frac{Q_1(\overline{d}_1)^{\oplus}, \ldots, Q_n(\overline{d}_n)^{\oplus}, Q_{n+1}(\overline{d}_{n+1})^{\ominus}, \ldots, Q_m(\overline{d}_m)^{\ominus}}{\top^{\ominus}}$$

is a *non-totality rule* for $\langle T, \tilde{I}\rangle$.
Note that the non-totality rule cannot be applied in case Δ is a total definition. Indeed, if Δ is total, then for every two-valued Open(Δ)-structure K, $\mathrm{wfm}_\Delta(K)$ is two-valued. It follows that when building a proof in a calculus that includes the non-totality rule, this rule should never be tried for positive, monotone or stratified definitions.

Soundness and completeness

The $\mathrm{MX}^{\mathrm{ID}}$-calculus for FO(ID) is the MX-calculus for FO, extended with the completion, unfounded set and non-totality rules. The results of sections 6.1.2 and 6.1.3 carry over to this extension of the MX-calculus. In particular, we have the following theorem.

Theorem 6.6. *There exists an MX^{ID}-proof for $\langle T, \tilde{I}\rangle$ containing only signed T-instances iff there is no model of T approximated by $\tilde{I}$.*

Proof. The proof is completely similar to the proof of Theorem 6.2. See Appendix A. ■

6.3.2 Debugging for FO(ID)

The debugging method for MX(FO) can be extended to a debugging method MX(FO(ID)), by constructing MX^{ID}-proofs instead of MX-proofs. There is no problem communicating the use of a completion rule to the user. For the unfounded set and non-totality rules however, this is not entirely straightforward. A possible explanation to a user who asks why $P(\overline{d})$ is false, where this atom was derived by an application of the unfounded set rule is (using the same notations as in the section where we introduced the unfounded set rule)

> Because $Q_1(\overline{d}_1)$, ..., $Q_n(\overline{d}_n)$ must be true and $Q_{n+1}(\overline{d}_{n+1})$, ..., $Q_m(\overline{d}_m)$ must be false, $P(\overline{d})$ belongs to a set U of defined atoms of Δ that can only become true because of circular reasoning.

If the user asks why the atoms in U can only become true because of circular reasoning, the system could let him explore a proof for the inconsistency of $\left\langle \bigvee_{\varphi[\overline{x}/\overline{d}]\in B} \varphi[\overline{x}/\overline{d}], \tilde{K}[U/\mathbf{f}] \right\rangle$, where the $B = \{\varphi[\overline{x}/\overline{d}] \mid \forall \overline{x}\ R(\overline{x}) \leftarrow \varphi[\overline{x}] \in \Delta \text{ and } R[\overline{x}/\overline{d}] \in U\}$. That is, a proof for the fact that $\tilde{K}[U/\mathbf{f}](\varphi[\overline{x}/\overline{d}])$ is false for any $\varphi[\overline{x}/\overline{d}] \in B$.
When the user asks why $\top$ is false, where $\top^{\ominus}$ was derived by an application of a non-totality rule, a possible explanation is (using the same notations as in the section where we introduced the non-totality rule)

> Because $Q_1(\overline{d}_1)$, ..., $Q_n(\overline{d}_n)$ must be true and $Q_{n+1}(\overline{d}_{n+1})$, ..., $Q_m(\overline{d}_m)$ must be false, and definition Δ of T is non-total with respect to K, T cannot have a model.

To explain the user why Δ is non-total with respect to K, the system could show the (construction of the) well-founded model of Δ extending K, and indicate that this model is strictly three-valued.

6.4 Related work

Much work in debugging for declarative programming systems focusses on a specific procedural semantics (e.g., on a particular execution model for Prolog). See, e.g., (Jahier et al., 2000; Langevine et al., 2003). The trace of a solver is then a sequence of steps according to the procedural semantics, rather than a formal proof. Details are given on how to obtain and store the trace efficiently. Tronçon and Janssens (2007) address these issues in a context where the small scope hypothesis does not hold.
Existing approaches for debugging an input for a model generator can be divided into two classes: the approaches that aim at *locating* a bug, and those that aim at *explaining* derivations made by a model generator. Clearly, these classes are complementary. A system of the first class can extract a part of the theory where the bug is located. Then, a system of the second class can explain why this part contains a bug. As far as we know, our debugging method is the first one for (extended) FO model generation that belongs to the second class.

The ALLOY system (Chang, 2007) is a model expansion system for a syntactic variant of FO. Shlyakhter et al. (2003a) present a debugging method for over-constrained (hence unsatisfiable) instances. Their method consists of extracting an *unsatisfiable core*, i.e., a small inconsistent subset, from the theory and presenting it to the user. Hence, it belongs to the first class. If the unsatisfiable core is small, the user can quickly locate the bug. If it is somewhat larger, it can still be difficult to detect the bug. In this case, our system could be used with the unsatisfiable core as input to further guide the search for a bug. This has the side benefit of speeding up our approach: a proof of inconsistency for the small unsatisfiable core is smaller and can be constructed faster than a proof for the inconsistency of the whole theory.

In the context of ASP, several approaches to debugging have been presented. A recent overview can be found in (Gebser et al., 2008). Most ASP debugging methods belong to the first class mentioned above. For instance, the method described by Gebser et al. (2008) returns for an input $\langle T, \tilde{I}\rangle$ a two-valued interpretation $M \geq_p \tilde{I}$ and a number of constraints, rules and/or unfounded sets that are violated by M. The method of (Syrjänen, 2006) returns a minimal set of rules such that the theory without these rules is satisfiable. An advantage of these two methods is that they can be implemented in ASP itself.

A debugging method of the second class for ASP was presented by Pontelli and Son (2006). It allows a user to interrupt the computation of an ASP solver and to ask an explanation for any atom that is not unknown at that moment. Explanations are given in the form of graphs, called *justifications*.

Another system worth mentioning is ASPVIZ (Cliffe et al., 2008), which can be used to inspect finite structures by visualizing them. Often, bugs can be detected easily on such a visual representation. ASPVIZ takes as input a finite structure I over a fixed vocabulary Σ_{draw}, and represents it by a two-dimensional drawing. For example, the vocabulary Σ_{draw} contains the predicate $\mathit{DrawRect}/4$ and if domain atom $\mathit{DrawRect}(\mathit{brush}, \mathit{Pt}(d_x, d_y), w, h)$ is true in input structure I, the system will draw a rectangle with upper left corner at position (d_x, d_y), width w and height h and the properties (thickness, colour, etc.) of brush. If one wants to visualize a finite structure J over a vocabulary Σ, one can use a theory over $\Sigma \cup \Sigma_{\text{draw}}$ describing which objects to draw. For example, to visualize a solution J of the battleship puzzle, a theory T is used containing, a.o., the definition

$$\left\{ \begin{array}{ll} \forall r \forall c\ (\mathit{DrawRect}(\mathit{BlackBrush}, \mathit{Pt}(r,c), 1, 1) & \leftarrow \mathit{ContainsShip}(r,c)) \\ \forall r \forall c\ (\mathit{DrawRect}(\mathit{BlueBrush}, \mathit{Pt}(r,c), 1, 1) & \leftarrow \neg \mathit{ContainsShip}(r,c)) \end{array} \right\}$$

Giving a model I for T expanding J to $\Sigma \cup \Sigma_{\text{draw}}$ to ASPVIZ will result in a drawing of a grid, where all squares containing a ship are black, and all others blue. We independently developed a system similar to ASPVIZ, called IDPDRAW. The system can be downloaded from `www.cs.kuleuven.be/~dtai/krr/software.html`. It helped to detect bugs in one of the checking programs for the second ASP competition.

A set of tableau calculi for ASP were described by (Gebser and Schaub, 2006). These calculi are closely related to the MX^{ID}-calculus. Specifically, the calculi

for ASP contain the conjunction, completion and cut rules of the MX^{ID}-calculus. The rules in the ASP tableau calculi to handle unfounded sets are tailored to describe the search of existing propositional ASP solver and are therefore different from the unfounded set rule of the MX^{ID}-calculus. The ASP calculi do not contain the non-totality, the disjunction, existential and universal rules.

6.5 Conclusion

In this chapter we presented a method for debugging logic theories by interactively investigating formal proofs that explain why there is no model of a theory expanding a certain structure. We showed that the formal proofs can be constructed by the constraint propagation based model generator we presented in Chapter 4.
It is part of future work to implement and evaluate our debugging method in practice. Since our method basically explains bugs, we expect that it will be a valuable addition to existing methods that aim at locating bugs.

Chapter 7

Model revision

Often, one is not (only) interested in a single solution to a search problem, but also in *revising* this solution under varying circumstances. For example, a network administrator is interested in computing an initial configuration of a computer network as well as in maintaining the configuration when one of the machines in the network breaks down. Typically, the following are requirements for a revised solution:

1. To allow a fast reaction on new circumstances, computing a revision should be efficient.

2. Executing a proposed revision in the problem domain usually has a certain cost. For example, it takes time to move mail servers from one computer to another. A good revision should preferably have a low cost. That is, it should be executable by a small number of cheap operations.

Most existing approaches to solve revision problems are tailored towards a specific application such as train rescheduling, and hence, they are not (entirely) declarative. In this chapter, we present a general, declarative revision method which can be applied if the original search problem can be represented as a model generation problem. Formally, we describe a revision problem by a theory T, a finite model M of T and a set of atoms C: T describes the original search problem, M a solution to that problem and C the atoms that should swap truth value compared to M. A solution to the revision problem is a model M' of T such that for every atom in C, its truth value in M' is opposite to its truth value in M. Revision problems where the theory T or the domain of M is changed can be reduced to this case, as we show in Section 7.1.2.
We consider revision problems where T is a first-order logic (FO) theory. Like finite model generation for FO, these problems can easily be reduced to SAT, i.e., to a propositional model generation problem, by grounding. An off-the-shelf SAT solver can then be used to tackle the latter problem. However, directly using this approach does not satisfy the two requirements mentioned above. It does not guarantee that the revised model is close to the original model and it

can take too much time and space to create and store the propositional theory. The method we propose avoids these problems by first constructing a set of atoms S', called a *search bound*. The bound contains all the atoms that we allow to swap truth value to obtain a revised model M' from the original one M. Then the method tries to find, by reducing to SAT, such a revised model M'. If it does not succeed, the bound S' is enlarged and the process is repeated. However, if it succeeds and S' is relatively small, the two requirements are met. Only a small number of operations should be performed to obtain M', because the only changes are on atoms in S'. Also, reducing to SAT can be efficient if S' is small.

To make the approach work, the consecutive search bounds should be constructed such that there is a *reasonable chance* that a revised model bounded by them exists. In Section 7.3, we present a non-deterministic algorithm to compute such search bounds. Experiments with a prototype implementation are discussed in Section 7.4. They indicate that the algorithm often finds small search bounds containing a revision.

7.1 The revision problem

For the rest of this chapter, we assume a fixed vocabulary Σ and finite domain D. Unless stated otherwise, every structure has domain D. Moreover, we assume Σ contains no function symbols and that in every theory, negations only occur directly in front of atoms. If I is a Σ-structure and R a set of domain atoms, we denote by $\mathrm{swap}(I, R)$ the structure obtained from I by swapping the truth values of the domain atoms in R. That is, $\overline{d} \in P^{\mathrm{swap}(I,R)}$ if either $\overline{d} \in P^I$ and $P(\overline{d}) \notin R$, or $\overline{d} \notin P^I$ and $P(\overline{d}) \in R$. We will often represent a finite two-valued structure by the set of all domain atoms that are true in it.

7.1.1 Basic revision problems

We now formally define the revision problem for FO. Let T be an FO theory, M a finite model of T and C a set of domain atoms. A *revision* for $\langle M, T, C\rangle$ is a set R of domain atoms such that $R \cap C = \emptyset$ and $\mathrm{swap}(M, R \cup C) \models T$. Intuitively, M describes the original solution to a problem, C the changes that occurred, and R the changes that should be made to repair the current situation $\mathrm{swap}(M, C)$. We call $\mathrm{swap}(M, R \cup C)$ a *revised model*.

Example 7.1. Consider the problem of placing n non-attacking rooks on an $n \times n$ chessboard. A model with domain $\{1, 2, \ldots, n\}$ of the following theory T_1 describes a solution to this problem. Here, atom $R(r, c)$ means "there is a rook

on square (r, c)".

$$\forall r\ \exists c\ R(r, c). \tag{7.1}$$
$$\forall c\ \exists r\ R(r, c). \tag{7.2}$$
$$\forall r \forall c_1 \forall c_2\ (R(r, c_1) \wedge R(r, c_2) \Rightarrow c_1 = c_2). \tag{7.3}$$
$$\forall r_1 \forall r_2 \forall c\ (R(r_1, c) \wedge R(r_2, c) \Rightarrow r_1 = r_2). \tag{7.4}$$

For $n = 3$, $M_1 = \{R(1,1), R(2,3), R(3,2)\}$ is a model. Assume we have computed M_1, but for some reason, we do not want a rook on position $(1,1)$. That means we have to search for a revision for $\langle T_1, M_1, C_1\rangle$, where $C_1 = \{R(1,1)\}$. A possible revision is the set $R_1 = \{R(1,3), R(2,1), R(2,3)\}$, which yields the revised model $\{R(1,3), R(2,1), R(3,2)\}$.

In practice, it is often the case that not all atoms are allowed to swap truth value in order to obtain a revised model. For example, in a train rescheduling problem, the truth value of atoms that state the positions of the railway stations should never change, because the position of the stations cannot be changed in reality. To formally describe such a problem, let S be a set of domain atoms, disjoint from C. Intuitively, S is the set of atoms that are allowed to swap truth value, i.e., the search space to find a revision. A revision R for $\langle T, M, C\rangle$ is *bounded* by S if $R \subseteq S$. The *bounded revision problem* with input $\langle T, M, C, S\rangle$ is the problem of finding a revision that is bounded by S.
A (bounded) revision problem with input $\langle T, M, C, S\rangle$ can easily be reduced to a model expansion problem. Indeed, R is a revision for $\langle T, M, C\rangle$ bounded by S iff $\text{swap}(M, R \cup C)$ is a solution to the model expansion problem with input $\langle T, \text{swap}(M, C)[S/\mathbf{u}]\rangle$. Vice versa, any model expansion problem with input $\langle T, \tilde{I}\rangle$ can be reduced to a model revision problem. To this end, introduce a new 0-ary predicate P and let T' be the theory containing the single sentence $P \Rightarrow (\bigwedge T)$. Let M' be a two-valued structure that assigns $\mathbf{f}$ to P and that is approximated by $\tilde{I}$. Let S' be the set of domain atoms that are unknown in $\tilde{I}$. Then R is a revision for $\langle T', M', \{P\}, S'\rangle$ iff $\text{swap}(M', R \cup \{P\})$ is a solution to the model expansion problem with input $\langle T, \tilde{I}\rangle$.
It follows that the computational complexity of model revision problem is equal to the computational complexity of model expansion. Hence the model revision problem is **NEXPTIME**-complete. If the input theory is fixed, the model revision problem is **NP**-complete (Mitchell and Ternovska, 2005).

7.1.2 Domain and theory changes

Besides the revision problem as described above, i.e., the problem of computing a new model when the truth values of some of the domain atoms change, one could also consider the problem of computing a new model when either[31]:

1. A new sentence is added to the theory.
2. A domain element is left out of the old model.

[31] Observe that it is trivial to compute a new model when a sentence of T is left out.

3. A new domain element is added to the old model.

All three problems can easily be reduced to the bounded revision problem defined above.
To reduce (1), let φ be the sentence that is added, let P be a new propositional atom and M a model of T such that P is false in M. Then a revision for $\langle T \cup \{P \Rightarrow \varphi\}, M, \{P\}\rangle$ is a model for $T \cup \{\varphi\}$. To reduce (2), let *Used* be a new unary predicate and denote by T' the theory obtained by replacing each subformula in T of the form $(\forall x\, \varphi)$, respectively $(\exists x\, \varphi)$, by $(\forall x\, (\mathit{Used}(x) \Rightarrow \varphi))$, respectively $(\exists x\, (\mathit{Used}(x) \wedge \varphi))$. Let U be the set of all domain atoms of the form $\mathit{Used}(d)$, M a model of T such that each atom in U is true in M, and denote by d' the domain element that is left out. A revision for $\langle T', M, \{\mathit{Used}(d')\}, \mathcal{H} \setminus U\rangle$, restricted to the atoms not mentioning d', is a model for T. Here, $\mathcal{H}$ denotes the set of all domains atoms over Σ and D, i.e., the Herbrand base. In a similar way, (3) can be reduced.

7.1.3 Weighted revision problems

In real-life revision problems, one is often more interested in a revision with a *low cost*, than in a small revision. For example, suppose some problem in a network can be solved by either moving multiple DHCP servers, or by moving only one mail server. Although the revision describing the first solution will have a higher cardinality than the second one, the cost of performing the second one is higher, since moving a mail server involves moving all mailboxes of users. Moving a DHCP server only involves copying a single script.
To model revision problems where the cost of the revision plays an important role, a pair (c_+, c_-) of positive numbers is assigned to each domain atom $P(\overline{d})$. These two numbers indicate the cost of changing the truth value of $P(\overline{d})$ from false to true, respectively from true to false. For instance, if $P(d)$ means that there is a mail server on machine d, c_+ indicates the cost of installing a mail server on d, while c_- indicates the cost of uninstalling it. A good revision only contains atoms that are false, respectively true, in the original model and are assigned a low c_+, respectively c_-.

7.2 Solving the revision problem

In the rest of this chapter, we assume a fixed T, M, C and S and consider the bounded revision problem for input $\langle T, M, C, S\rangle$. The problem $\langle T, M, C, S\rangle$ can be solved by reducing it to the problem of finding a model of a propositional theory T_g, called a *grounding*. The model generation problem can then be solved by an off-the-shelf efficient SAT solver (Mitchell, 2005). To find revisions with low cardinality (or low cost in case of a weighted revision problem), one can use a SAT solver with support for optimization, such as a max-SAT solver (Li et al., 2007; Pipatsrisawat and Darwiche, 2007).

7.2.1 Grounding

Recall that a formula φ is in *ground normal form (GNF)* if it is a boolean combination of domain atoms. That is, φ is a sentence containing no quantifiers or variables. A GNF theory is a theory containing only GNF formulas. As noted before, a GNF theory is essentially a propositional theory.

Definition 7.1. A *grounding for* $\langle T, M, C, S\rangle$ is a GNF theory T_g such that for every $R \subseteq S$, R is a revision for $\langle T, M, C, S\rangle$ iff $\text{swap}(M, C \cup R) \models T_g$. A grounding is called *reduced* if all domain atoms that occur in it belong to S.

The grounding algorithm of Chapter 5 (Algorithm 5.1) can be used to compute groundings for revision problems. A grounding for the revision problem $\langle T, M, C, S\rangle$ is a grounding for the model expansion problem with input theory T and the three-valued input structure $\text{swap}(M, C)[S/\mathbf{u}]$. We indicated in Section 5.5 how our grounding algorithm (Algorithm 5.1) can be adapted to take three-valued structures as input.

7.2.2 Optimized grounding

If the search space S is small, a grounder that produces a reduced grounding spends most of its time evaluating formulas in $\text{swap}(M, C)[S/u]$. According to Theorem 2.1, evaluating a formula φ in a structure takes polynomial space in the size of φ. However, by exploiting the fact that M is a model of T, there are certain subformulas of T that can be evaluated in constant time. Below, we describe a set of subformulas of T with this property. Intuitively, the set contains formulas that are obviously true in M, and cannot but remain true in $\text{swap}(M, C)$.
First, we introduce a set of subformulas of T that are obviously true in M. Inductively define the set $\tau(T)$ by

- If φ is a sentence of T, then $\varphi \in \tau(T)$;
- If $\varphi_1 \wedge \varphi_2 \in \tau(T)$, then $\varphi_1 \in \tau(T)$ and $\varphi_2 \in \tau(T)$;
- If $\forall x\ \varphi[x] \in \tau(T)$, then $\varphi[x] \in \tau(T)$.

If $\varphi[\overline{x}]$ is a formula of $\tau(T)$, then clearly $M\theta \models \varphi[\overline{x}]$ for every variable assignment θ. Note that the set $\tau(T)$ can be constructed in time linear in the size of T.
Next, we investigate which of the formulas of $\tau(T)$ remain true in every revised model. The set of formulas from $\tau(T)$ that do not contain *dangerous literals* is an underestimation of the set of formulas that remain true.

Definition 7.2. A domain literal L is *dangerous in a formula* φ *with respect to* S if $\text{atom}(L) \in C \cup S$, $M \models L$ and there exists a tuple $\overline{d}$ and a literal $L'[\overline{x}]$ that occurs positively in φ such that $L = L'[\overline{x}/\overline{d}]$.

Intuitively, a literal is dangerous in φ if it has a negative influence on the truth value of φ in $\text{swap}(M, C \cup S)$, while it had a positive influence in M. We denote the set of all dangerous literals in φ with respect to S by $\mathcal{D}_S(\varphi)$.

Lemma 7.1. *If* $\mathcal{D}_S(\varphi) = \emptyset$*, then* $\mathrm{swap}(M, C \cup R)\theta(\varphi) \geq_t M\theta(\varphi)$ *for every* $R \subseteq S$ *and every variable assignment* θ*.*

Proof. If φ is a literal $L[\overline{x}]$, then it follows from the assumption $\mathcal{D}_S(\varphi) = \emptyset$ that for every tuple of domain elements $\overline{d}$, either $M \not\models L[\overline{d}]$ or $L[\overline{d}] \notin (C \cup S)$. If $M \not\models L[\overline{d}]$, then for every structure M', including structures of the form $\mathrm{swap}(M, C \cup R)$, $M'(L[\overline{d}]) \geq_t M(L[\overline{d}])$. If on the other hand $L[\overline{d}] \notin (C \cup S)$, then $\mathrm{swap}(M, C \cup R)(L[\overline{d}]) = M(L[\overline{d}])$ for every $R \subseteq S$. We conclude that $\mathrm{swap}(M, C \cup R)\theta(L[\overline{x}]) \geq_t M\theta(L[\overline{x}])$ for any variable assignment θ and $R \subseteq S$. The cases where φ is not a literal follow by induction. ■

Corollary 7.2. *If* $\varphi[\overline{x}] \in \tau(T)$ *and* $\mathcal{D}_S(\varphi[\overline{d}]) = \emptyset$*, then* $\mathrm{swap}(M, C \cup R) \models \varphi[\overline{d}]$ *for every* $R \subseteq S$ *and tuple of domain elements* $\overline{d}$*.*

It follows that a grounder can safely substitute formulas that satisfy the conditions of Corollary 7.2 by $\top$. Checking these conditions for a formula φ takes only linear time in the size of φ. In a grounder using bounds, this can be accomplished as follows. For an input $\langle T, M, C, S\rangle$ over vocabulary Σ, let σ be the vocabulary consisting of two predicates P_M/n and P_{ps}/n for each predicate $P/n \in \Sigma$. Let I_σ be a σ- structure defined by $\overline{d} \in P_M^{I_\sigma}$ iff $\overline{d} \in P^M$ and $\overline{d} \in P_{\mathrm{ps}}^{I_\sigma}$ iff $P(\overline{d}) \in C \cup S$. That is, P_M denotes the interpretation of P in M and P_{ps} denotes the domain atoms that are certainly or possibly swapped in a revision. Introduce for each positive literal $P(\overline{x})$ the formula $\nu_{P(\overline{x})}$ over σ defined by

$$P_M(\overline{x}) \Rightarrow \neg P_{\mathrm{ps}}(\overline{x}).$$

For each negative literal $\neg P(\overline{x})$, let $\nu_{\neg P(\overline{x})}$ be the formula

$$\neg P_M(\overline{x}) \Rightarrow \neg P_{\mathrm{ps}}(\overline{x}).$$

Now we compose a ct-bound φ_{ctb} for each of the formulas $\varphi \in \tau(T)$.

Lemma 7.3. *Let* $\varphi[\overline{x}] \in \tau(T)$ *and let* $L_1, \ldots, L_n$ *be all literals that occur positively in* φ*. Denote by* $\overline{y}$ *the variables that occur free in* $L_1, \ldots, L_n$ *but do not belong to* $\overline{x}$*. Then* $\forall \overline{y}\ (\nu_{L_1} \wedge \ldots \wedge \nu_{L_n})$ *is a ct-bound for* φ *over* σ*.*

Proof. Recall that a solution to the revision problem with input $\langle T, M, C, S\rangle$ is a model for T approximated by $\mathrm{swap}(M, C)[S/\mathbf{u}]$. Denote by $\tilde{I}$ the union of the structures $\mathrm{swap}(M, C)[S/\mathbf{u}]$ and the structure I_σ defined above. We have to show that $T \models_{\tilde{I}} \forall \overline{x}(\forall \overline{y}\ (\nu_{L_1} \wedge \ldots \wedge \nu_{L_n}) \Rightarrow \varphi[\overline{x}])$. Let J be a structure approximated by $\tilde{I}$ and let $\overline{d}_x$ be a tuple of domain elements such that $J[\overline{x}/\overline{d}_x] \models \forall \overline{y}\ (\nu_{L_1} \wedge \ldots \wedge \nu_{L_n})$. We have to show that $J \models \varphi[\overline{d}_x]$. First observe that $\mathcal{D}_S(\varphi[\overline{d}_x]) = \emptyset$. Indeed, if a literal $L_i[\overline{x}, \overline{y}]$ that occurs positively in φ is true in M, then for every tuple $\overline{d}_y$, the atom $\mathrm{atom}(L_i[\overline{x}/\overline{d}_x, \overline{y}/\overline{d}_y])$ does not belong to $C \cup S$, since $J[\overline{x}/\overline{d}_x, \overline{y}/\overline{d}_y] \models \nu_{L_i}$. From Corollary 7.2, we conclude that $\mathrm{swap}(M, C \cup R) \models \varphi[\overline{d}_x]$ for every $R \subseteq S$. Therefore $J \models \varphi[\overline{d}_x]$. ■

Example 7.2 (Example 7.1 ctd.)**.** Consider the theory T_1 from Example 7.1. The subformula $\exists c\ R(r,c)$ of sentence (7.1) belongs to $\tau(T_1)$. The only literal that occurs positively in this formula is the literal $R(r,c)$. It follows that $\forall c\ (R_M(r,c) \Rightarrow \neg R_{\text{ps}}(r,c))$ is a ct-bound for $\exists c\ R(r,c)$. Intuitively, this ct-bound states that if each rook on row r cannot be moved, there is still a rook on row r in the revised model. It follows that instead of grounding $\exists c\ R(r,c)$ for each row r, it is sufficient to only ground it for the rows containing a rook that can possibly be moved.
The subformula $R(r,c_1) \wedge R(r,c_2) \Rightarrow c_1 = c_2$ of sentence (7.3) also belongs to $\tau(T_1)$. The three literals that occur positively in this formula are $\neg R(r,c_1)$, $\neg R(r,c_2)$ and $c_1 = c_2$. Hence, the formula

$$(\neg R_M(r,c_1) \Rightarrow \neg R_{\text{ps}}(r,c_1)) \wedge (\neg R_M(r,c_2) \Rightarrow \neg R_{\text{ps}}(r,c_2))$$

is a ct-bound for $R(r,c_1) \wedge R(r,c_2) \Rightarrow c_1 = c_2$. It states that if no extra rooks can be placed on a row r, there is still at most one rook on that row.

7.3 Reducing the search space

Directly solving a revision problem $\langle T, M, C, S\rangle$ by reducing it to SAT has the drawback that there is no guarantee that the revised model that is found is *close* to M, i.e., that the cardinality of the corresponding revision is low. The revision R can be as large as S. On the other hand, one can often find a revision with a cardinality much lower than the number of domain atoms that are allowed to swap truth value. For example, in a network configuration problem, a problem in a certain subnet can often be repaired by only making changes within that subnet, not touching the rest of the network. In Example 7.1, the revision contains three atoms, while there are $n \times n$ domain atoms in total.

7.3.1 Search bounds

The observation above suggests the following approach to find a revision for $\langle T, M, C, S\rangle$. First, construct a *small* subset S' of S such that there might exist a revision for $\langle T, M, C, S'\rangle$. We call S' the *search bound*. Then, try to find a revision for $\langle T, M, C, S'\rangle$ by reducing to SAT. If a revision R is found, this is also a revision for $\langle T, M, C, S\rangle$, because $R \subseteq S' \subseteq S$. If, on the other hand, there is no revision for $\langle T, M, C, S'\rangle$, the search bound S' is enlarged. This process is repeated until a revision is found or $S' = S$. An example run of this algorithm for Example 7.1 is shown in Figure 7.1, where the grey squares represent the domain atoms in the consecutive search bounds. There are several important benefits of using this approach compared to directly reducing to SAT:

- If a small search bound S' can be detected such that there is a revision for $\langle T, M, C, S'\rangle$, such a revision is small. Hence, the corresponding revised model is close to the original model M.

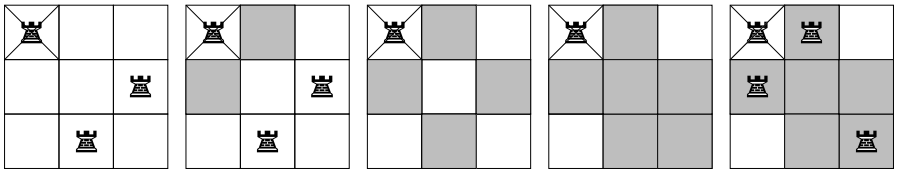


Figure 7.1: Enlarging the search bound

- The size of the reduced grounding for $\langle T, M, C, S' \rangle$ is small when S' is small. Indeed, all atoms that occur in the reduced grounding are atoms of S'. In general, the smaller the grounding, the faster the SAT solver can produce its answer.
- A small S' might speed up the grounding considerably, since the number of formulas that satisfy the conditions of Corollary 7.2 depends on the size of S'.

7.3.2 Enlarging the search bound

Assume that we have constructed a search bound S' that is not large enough. That is, there is no revision for $\langle T, M, C, S' \rangle$. Corollary 7.2 implies that this problem is caused by the dangerous literals in the sentences of T with respect to S'. If we want to enlarge S' to a new search bound S'' such that there is a reasonable chance that there is a revision for $\langle T, M, C, S'' \rangle$, we should add atoms to S' that can have a *positive influence* on the sentences of T. That is, if they swap truth value compared to M, then some literals that occur positively in T become true. More precisely:

Definition 7.3. A domain atom A has *positive*, respectively *negative*, *influence* on a formula φ if there is a positive occurrence of a literal $L[\overline{x}]$ in φ such that for some tuple $\overline{d}$, $\mathrm{atom}(L[\overline{x}/\overline{d}]) = A$ and $M \not\models L[\overline{x}/\overline{d}]$, respectively $M \models L[\overline{x}/\overline{d}]$. A domain atom has *positive influence on theory* T if it has positive influence on at least one sentence of T.

Note that if a domain literal L is dangerous in φ with respect to a search bound S', then $\mathrm{atom}(L)$ has negative influence on φ.
The following lemma illustrates that swapping the truth value of an atom with positive influence and no negative influence on a formula increases the truth value of that formula. The lemma can be proven by a straightforward proof by structural induction.

Lemma 7.4. *If domain atom A has positive influence and no negative influence on formula φ, then* $\mathrm{swap}(M, \{A\})\theta(\varphi) \geq_t M\theta(\varphi)$ *for every variable assignment θ. Vice versa, if A has negative influence but no positive influence on φ, then* $\mathrm{swap}(M, \{A\})\theta(\varphi) \leq_t M\theta(\varphi)$.

In most cases, it is not beneficial to choose the atoms to add to S' randomly among the atoms with positive influence on T. Rather, atoms that are added should be able to *neutralize* the negative influence of the dangerous literals. For example, if a literal $L[\overline{x}/\overline{d}]$ is dangerous in formula $\psi_1 \vee \psi_2$ with respect to S' and $L[\overline{x}]$ occurs in ψ_1, then its negative influence can be neutralized by making ψ_2 true. Hence in this case, one should add atoms to S' that have a positive influence on ψ_2. Observe that a dangerous occurrence of a literal L in a conjunction $\psi_1 \wedge \psi_2$ cannot be neutralized. If ψ_1 is false, then even if ψ_2 is true, the conjunction is false. These intuitions are formalized in the following definitions.

Definition 7.4. An instance ψ of a disjunctive subformula of T is a *neutralizable sentence for domain literal* L if $L \in \mathcal{D}_{S'}(\psi)$ and $\mathrm{swap}(M, C)[S'/\mathbf{u}](\psi) \leq_t \mathbf{u}$.

The condition $\mathrm{swap}(M, C)[S'/\mathbf{u}](\psi) \leq_t \mathbf{u}$ expresses that ψ might become false when some of the atoms in $S' \cup C$ swap truth value. That is, ψ can be a reason that there is no revision bounded by S'.

Definition 7.5. Let ψ be a neutralizable sentence for domain literal L. A domain atom A is a *neutralizer for* L *in* ψ if one of the following holds:

- $\psi = \bigvee_{1 \leq i \leq n} \chi_i$, $L \in \mathcal{D}_{S'}(\chi_j)$ and A has positive influence in χ_k for some $k \neq j$.
- $\psi = \exists x\ \chi[x]$, $L \in \mathcal{D}_{S'}(\chi[x/d])$ and A has positive influence in $\chi[x/d']$ for some domain element $d' \neq d$.

Example 7.3. Let φ be the propositional formula $((P \vee Q) \wedge R) \vee (U \wedge V)$ and let $M = \{P, R, V\}$, $C = \{P\}$ and $S' = \emptyset$. Then P is dangerous in φ. Its negative influence is clear: $M \models \varphi$, but $\mathrm{swap}(M, C \cup S') \not\models \varphi$. There are two neutralizable sentence for P: $P \vee Q$ and φ itself. Q is a neutralizer for the former, U for the latter. Observe that both $\mathrm{swap}(M, C \cup \{Q\})$ and $\mathrm{swap}(M, C \cup \{U\})$ satisfy φ. This suggests to add Q and/or U to S' to obtain a new search bound.

Example 7.4 (Example 7.1 ctd.)**.** Let $S' = \{R(2, 1)\}$. $R(1, 1)$ is dangerous in $\forall r \exists c\ R(r, c)$. The only neutralizing sentence in this case is $\exists c\ R(1, c)$, yielding $R(1, 2)$ and $R(1, 3)$ as neutralizers. $\neg R(2, 1)$ is dangerous in $\forall r \forall c_1 \forall c_2\ (R(r, c_1) \wedge R(r, c_2) \Rightarrow c_1 = c_2)$. $(R(2, 1) \wedge R(2, 3) \Rightarrow 1 = 3)$ is a neutralizable sentence for $\neg R(2, 1)$, with $R(2, 3)$ as neutralizer.

Our algorithm to enlarge the search bound consists of first computing a non-empty set V of neutralizers for literals in $\mathcal{D}_{S'}(T)$ such that $V \subseteq S$ and $V \cap S' = \emptyset$. Then, $V \cup S'$ is used as the next search bound.

As a summary, Algorithm 7.1 presents the complete algorithm for computing revisions. For input $\langle T, M, C, S \rangle$, the search bound S' is initially the empty set (line 1). In lines 3–6, grounding and propositional model generation are applied to find a revision for $\langle T, M, C, S' \rangle$. If such a revision does not exists, the search bound S' is enlarged (lines 7–12). First, the set N of all neutralizers that do not yet belong to S' is computed. If this set is empty, it is concluded that

Algorithm 7.1: Computing revisions

Input: A theory T, model M of T and disjoint sets of domain atoms C and S.

Output: A revision for $\langle T, M, C\rangle$, bounded by S, if such a revision exists. Otherwise, FAIL is returned.

```
1  S' := ∅;
2  while true do
3      T_g := ground(⟨T, M, C, S'⟩);
4      if T_g is satisfiable then
5          M' := a model of T_g;
6          return {A ∈ S' | M(A) ≠ M'(A)};
7      else
8          N := {A ∈ (S \ S') | A is a neutralizer for a literal L ∈ D_{S'}(T)};
9          if N = ∅ then return FAIL;
10         else
11             Choose a non-empty subset V of N;
12             S' := S' ∪ V;
```

no revision for $\langle T, M, C, S\rangle$ exists. Else, a subset V of N is chosen and S' is updated to $S' \cup V$.

Because S' increases each time the while loop is executed, and $S' \subseteq S$, it follows that Algorithm 7.1 terminates. The correctness of the algorithm follows from the next proposition.

Proposition 7.5. *Let $S' \subseteq S$ and let N be the set*

$$\{A \in (S \setminus S') \mid A \text{ is a neutralizer for a literal } L \in \mathcal{D}_{S'}(T)\}.$$

If there is no revision for $\langle T, M, C, S'\rangle$ and $N = \emptyset$, then there is no revision for $\langle T, M, C, S\rangle$.

Proof. Assume there is no revision for $\langle T, M, C, S'\rangle$ and there exists a revision R for $\langle T, M, C, S\rangle$. We have to show that $N \neq \emptyset$. Because there is no revision for $\langle T, M, C, S'\rangle$, there exists a sentence φ of T such that $\text{swap}(M, C \cup (R \cap S')) \not\models \varphi$. We now show by structural induction that either $\text{swap}(M, C \cup R) \not\models \varphi$ or $N \neq \emptyset$. Since R is a revision, it follows that the former case is impossible, and therefore $N \neq \emptyset$, as desired.

For the base case of the induction, assume φ is a literal L. Because $M \models L$ and $\text{swap}(M, C \cup (R \cap S')) \not\models L$, it follows that $\text{atom}(L) \in (C \cup R)$. Hence $\text{swap}(M, C \cup R) \not\models \varphi$.

Now assume φ is the conjunction $\psi_1 \wedge \psi_2$. Then $M \models \psi_1$ and $M \models \psi_2$. Also, $\text{swap}(M, C \cup (R \cap S')) \not\models \psi_1$ or $\text{swap}(M, C \cup (R \cap S')) \not\models \psi_2$. Without loss of generality, we assume $\text{swap}(M, C \cup (R \cap S')) \not\models \psi_1$. By the induction hypothesis, it follows that either $\text{swap}(M, C \cup R) \not\models \psi_1$ or $N \neq \emptyset$. The case where φ is of the form $\forall x\, \psi[x]$ is similar.

Finally, assume φ is the formula $\psi_1 \vee \psi_2$. Because $\text{swap}(M, C \cup (R \cap S')) \not\models \varphi$, it follows that $\text{swap}(M, C \cup (R \cap S')) \not\models \psi_1$. If both M and $\text{swap}(M, C \cup R)$ satisfy ψ_1, the induction hypothesis guarantees that $N \neq \emptyset$. Similarly if M and $\text{swap}(M, C \cup R)$ both satisfy ψ_2. Now assume M satisfies ψ_1 but not ψ_2, and $\text{swap}(M, C \cup R)$ satisfies ψ_2 but not ψ_1. Because $\text{swap}(M, C \cup (R \cap S')) \not\models \psi_1$, it follows from Lemma 7.1 that $\mathcal{D}_{S'}(\psi_1) \neq \emptyset$ and hence $\mathcal{D}_{S'}(\varphi) \neq \emptyset$. Let L be a domain literal that belongs to $\mathcal{D}_{S'}(\psi_1)$. Since $M \models \varphi$, $\text{swap}(M, C \cup (R \cap S')) \not\models \varphi$ and $\text{swap}(M, C \cup R) \models \varphi$, it must be the case that $\text{swap}(M, C)[S'/u](\varphi) = \mathbf{u}$. Hence φ is a neutralizable sentence for L. To prove that $N \neq \emptyset$, it now suffices to show that there exists a neutralizer for L in φ. Because $\text{swap}(M, C \cup R) \models \psi_2$ while $\text{swap}(M, C \cup (R \cap S')) \not\models \psi_2$, it follows that there exists a literal $L'[\overline{x}]$ that occurs positively in ψ_2 and a tuple of domain elements $\overline{d}$ such that $\text{swap}(M, C \cup (R \cap S')) \not\models L'[\overline{x}/\overline{d}]$ and $\text{atom}(L'[\overline{x}/\overline{d}]) \in (R \setminus S')$. Hence $M \not\models L'[\overline{x}/\overline{d}]$. Therefore $\text{atom}(L'[\overline{x}/\overline{d}])$ has positive influence on ψ_2. We conclude that $\text{atom}(L'[\overline{x}/\overline{d}])$ is a neutralizer for L in φ. The case where φ is of the form $\exists x\ \psi[x]$ is similar. ■

In the above, we did not specify how to choose the set of neutralizers V among the neutralizers in N to enlarge the search bound (line 11 in Algorithm 7.1). Several heuristics can be thought of. A thorough theoretical and experimental investigation of heuristics is part of future work. We implemented the following simple heuristics, but none of them produced significantly better results than making random choices:

- In order to neutralize multiple dangerous literals at once, prefer to add atoms to S' that are a neutralizer for more than one literal in $\mathcal{D}_{S'}(T)$.
- Prefer to add atoms to S' that have negative influence on only few subformulas of T. This minimizes the number of dangerous literals in the next iteration of the while loop.
- A weighted sum of the two heuristics above.

A more sophisticated heuristic, based on the trace of the SAT solver applied in line 4 of Algorithm 7.1, was investigated by De Cat (2009). Also this heuristic did not produce significantly better results than randomly choosing a subset V from N. In the experiments of the next section, all results are obtained with random heuristics.

7.4 Implementation and experiments

We made a prototype implementation of Algorithm 7.1 on top of the model generator IDP. As mentioned, we implemented line 11 by choosing a random subset V of the neutralizers for literals in $\mathcal{D}_{S'}(T)$ (the set N in Algorithm 7.1). The grounding phase (line 3) was implemented as described in Section 7.2.2. We tested the implementation on three different problems.

N-Rooks: The N-Rooks puzzle as described in Example 7.1. For each of the instances, C contains three atoms. That is, at least three atoms swap truth value compared to the given model.

N-Queens: The classical N-Queens puzzle. C contains exactly one atom for each of the instances.

$M \times N$-Queens: An extension of the N-Queens puzzle. Here, M chess-boards of dimension $N \times N$ are placed in a circle. On each board, a valid N-Queens solution is computed. Moreover, if a board contains a queen on square (x, y), then the neighbouring boards may not contain a queen on (x, y). C contains one atom for each of the instances. In this case, it often suffices to revise one of the boards to revise the whole problem.

In the tables below, *MG* stands for *model generation*. Times and sizes in columns marked *MG* denote times and sizes obtained when solving the problem from scratch with the IDP system. *MR* stands for *model revision*. Numbers in these columns are obtained with our revision system. All sizes are expressed in the number of propositional literals. The grounding size for model revision is the grounding size for the final search bound. The *full bound size* is the size of the entire search space, the *final bound size* is the average size of the final search bound. *Iter* denotes the average number of iterations, *Changes* the average number of changes with respect to the original model. We used a time-out (###) of 600 seconds. All results are averaged over 10 runs. In each of the experiments, the cardinality of the randomly chosen subset V of N was restricted to a maximum of 10 domain atoms.
On the N-rooks problems, the revision algorithm scores well. Up to dimension 5000 is solved within 30 seconds, while model generation was impossible for dimensions above 500. Around 3 iterations are needed to find a sufficiently large search bound. This is as expected, since this problem is not strongly constrained: it is possible to find revisions of low cardinality. On average, 5 rooks were moved to obtain a revised model.
The results on the N-Queens show the weakness of the random heuristic on strongly constrained problems. Model revision turns out to be far slower than generating a new model from scratch. Also, many queens are moved to obtain the revised model. For instance, for dimension 40 and 50, 15 queens are moved.
The $M \times N$-queens problems show the strength of the revision algorithm. For problems consisting of a large number of small subproblems, the algorithm is able to revise an affected subproblem without looking at the other subproblems. On problems with a large number of boards, this results in better times compared to model generation.

7.5 Related and future work

Most literature on revision of solutions focusses on particular applications such as train rescheduling. We are not aware of papers describing techniques to

Table 7.1: N-Rooks

	Time (sec.)		Grounding size		Bound size			
	MG	MR	MG	MR	Full	Final	Iter.	Changes
10	0	1	$3.8 \cdot 10^3$	585	100	28	2.8	10.0
50	0	1	$5.0 \cdot 10^5$	479	2500	24	2.4	10.4
100	2	1	$3.9 \cdot 10^6$	698	10000	28	2.8	10.4
250	30	1	$6.2 \cdot 10^7$	903	62500	29	2.9	10.6
500	###	1	$5.0 \cdot 10^8$	1116	$2.5 \cdot 10^5$	28	2.8	10.4
1000	###	2	###	1998	$1.0 \cdot 10^6$	28	2.8	11.0
5000	###	29	###	5575	$2.5 \cdot 10^7$	26	2.6	11.0

Table 7.2: N-Queens

	Time (sec.)		Grounding size		Bound size			
	MG	MR	MG	MR	Full	Final	Iter.	Changes
10	0	1	3140	1943	100	83	8.3	14
20	0	2	25880	6133	400	93	9.3	14
30	0	7	88220	16690	900	121	12.1	26
40	1	12	210160	30083	1600	135	13.5	30
50	2	24	411700	49317	2500	149	14.9	30
60	4	###	712840	###	3600	###	###	###

Table 7.3: $M \times N$-Queens

	Time (sec.)		Grounding size		Bound size			
	MG	MR	MG	MR	Full	Final	Iter.	Ch.
10×10	0	2	33400	4387	1000	95	10.1	14
100×10	1	5	334000	5932	10000	127	13.3	17
1000×10	###	29	3340000	5932	100000	127	14.6	17
10×25	1	58	521000	34872	6250	408	41.4	25
100×25	178	104	5210000	39540	62500	475	48.1	27
1000×25	###	277	52100000	39540	625000	475	44.6	27

handle the general revision problem for FO described in this chapter. In work dealing with propositional logic, e.g., by Fox et al. (2006), the heuristics take large parts of the propositional theory into account, which becomes infeasible for problems with large domains.

In the area of Answer Set Programming (ASP), the problem of model revision was described by Brewka (2004).[32] He indicates that model revision problems can be solved by applying an answer set solver with support for soft constraints, i.e., for optimization. This corresponds to what we explained in Section 7.1.1.

Brain et al. (2005) present a method for updating answer sets of a logic program when a rule is added to it. The method is completely different from the one we presented: it does not rely on existing ASP or SAT solvers, it cannot handle the case where a rule is removed from the program and it works on the propositional level.

The problem of repairing inconsistent databases with integrity constraints (e.g., Arieli et al., 2004; Bertossi and Schwind, 2004) consists of finding a model of a (restricted) FO theory T that is as close as possible to a given structure I. This structure I is often the result of integrating different databases. On the contrary, the structure $\text{swap}(M, C)$ that needs to be revised in the model revision problem, is the result of swapping the truth of atoms in a *model* of T. The methods we presented to solve model revision problems critically depend on the fact that M is a model of T and therefore cannot directly be applied for repairing databases. Vice versa, the methods described in the literature to solve database repair problems are not tuned towards model revision.

Instead of using our method with a DPLL based SAT solver as back-end, one could directly use a local search SAT solver (Selman et al., 1993) on the grounding of $\langle T, M, C, S\rangle$ and provide the original model M as starting point for the search. The main difference with our approach is the way conflicts (dangerous literals) are handled. A local search solver immediately swaps the truth value of neutralizers and hence maintains a two-valued structure, while our approach implicitly assigns the truth value *unknown* to all atoms in the search bound and hence maintains a three-valued structure.

The following are topics for future work:

- A thorough theoretical and experimental study of heuristics. A promising approach is to analyze the proof of unsatisfiability produced by the SAT solver on an input $\langle T, M, C, S'\rangle$. This analysis then guides the extension of the search bound S'. Also heuristics applied in local search SAT and constraint solvers could be of use. A different approach consists of developing a more interactive system where the user can guide the search.

- The current implementation spends most of its time in grounding. For two search bounds $S' \subseteq S''$, the grounding for $\langle T, M, C, S'\rangle$ is a subtheory of the grounding for $\langle T, M, C, S''\rangle$. As such the same subtheory is grounded multiple times. It is part of future work on the implementation to avoid this overhead by grounding in an incremental way.

[32] Model revision is called *solution coherence* in Brewka's paper.

- We plan to extend the revision algorithm to FO(·).

7.6 Conclusion

In this chapter we defined the model revision problem for logic theories, the problem of computing a new model of a theory when certain changes to an existing model are required. We showed that several interesting variants of the revision problem can be reduced to the basic revision problem. Furthermore, we outlined a simple grounding-based method to solve revision problems and investigated how this method can be improved by restricting the search space for finding a revised model. If no revised model is found, the search space is gradually increased. Preliminary results obtained with a prototype implementation indicate that our method is often capable of computing revised models that are slight modifications of the given original model.

Chapter 8

Conclusions

We started this thesis by mentioning one of the long-term goals of research in the field of Knowledge Representation and Reasoning, namely the development of a knowledge base system (KBS). In such a system, knowledge is stored by specifying it in a suitable logic. Then, different tasks are solved by applying off-the-shelf reasoning systems on the stored knowledge.

This thesis contributes to the development of a KBS in several ways. In Chapter 3, we combined different language constructs, most of them proposed earlier in the literature, with classical logic to form the logic FO($\cdot$). We argued that FO($\cdot$) often allows for more concise and natural specifications of problems domains than possible in FO. Like classical logic, FO($\cdot$) has a clear formal semantics and a corresponding informal semantics. In other words, the intuitive reading of a formula in FO($\cdot$) corresponds to the mathematically defined meaning of this formula. This property is of prime importance for a logic underlying a KBS, as it implies a user can safely rely on the intuitive reading when specifying or checking the correctness of knowledge in FO($\cdot$).

In Chapters 4, 5 and 7, we defined and studied several forms of inference that are useful for solving real-life computational tasks. Constraint propagation serves to derive from a given theory and incomplete information more complete information. We indicated that constraint propagation can directly be applied to implement knowledge based configuration systems and finite model generators. Indirectly, it can be used to improve grounding and approximate query answering. As such, the method for (symbolic) constraint propagation we presented contributes to the development of a KBS. In Chapters 5 and 7 we showed that grounding can in turn be applied to implement part of model generation and model revision systems.

The method for debugging theories, presented in Chapter 6, contributes to the goal of building a practical KBS since it can be used as a tool to facilitate writing theories. Also, we expect that the method can be of use in a configuration system. Indeed, if a user of a configuration system tries to make a selection that violates the given constraints, our debugging method can be applied to explain the user why his or her selection is impossible.

There are still many aspects of a KBS that need to be investigated before a successful system can be built. Some of these issues are of a theoretical nature. Further optimization of the algorithms that are currently used to implement reasoning tasks and proposing and studying new language constructs and new forms of inference are such aspects. The topics for future work mentioned in the different chapters of this thesis belong to this category.
Other issues in the development of a real-life KBS are of a more practical nature. For instance, classical logic is believed to be too complicated to be an acceptable specification language in the industry. A fortiori, the same applies for FO($\cdot$). A workaround is to propose an acceptable syntactic variant of FO($\cdot$) as specification language, a solution that proved to be successful in database management systems. Another practical aspect of building a KBS is the development of a suitable user interface and/or the embedding of KBS "instructions" as a library in existing programming languages. The success of systems like KODKOD and ALLOY indicate the importance of such an embedding, respectively interface.

Appendix A

Proofs

This chapter contains, for the sake of completeness, the simple but tedious proofs that were omitted from the main chapters in the thesis.

Proof of Proposition 3.1

Lemma A.1. *Let φ be the atom $P(t_1, \ldots, t_n)$ or $F(t_1, \ldots, t_{n-1}) = t_n$ and let $f_1, \ldots, f_m$ be all ambiguous subterms of $(t_1, \ldots, t_n)$. Let $x_1, \ldots, x_m$ be new variables. Define by induction for every subterm t of φ the term t' by*

$$t' = \begin{cases} t & \text{if } t \text{ is a variable} \\ x_i & \text{if } t = f_i \\ G(g'_1, \ldots, g'_k) & \text{if } t = G(g_1, \ldots, g_k) \text{ and } t \neq f_i \text{ for every } 1 \leq i \leq m \end{cases}$$

If f_i is the term $G_i(g_1, \ldots, g_k)$ then denote by Q''_i the atom $\mathscr{G}_{G_i}(g'_1, \ldots, g'_k, x_i)$. Every cautious rewrite of $P(t_1, \ldots, t_n)$, respectively $F(t_1, \ldots, t_{n-1}) = t_n$ is equivalent to

$$\forall x_1 \cdots \forall x_m \ (Q''_1 \wedge \ldots \wedge Q''_m \Rightarrow P(t'_1, \ldots, t'_n)), \tag{A.1}$$

respectively

$$\forall x_1 \cdots \forall x_m \ (Q''_1 \wedge \ldots \wedge Q''_m \Rightarrow \mathscr{G}_F(t'_1, \ldots, t'_n)). \tag{A.2}$$

Every cautious rewrite of $\neg P(t_1, \ldots, t_n)$, respectively $F(t_1, \ldots, t_{n-1}) \neq t_n$, is equivalent to

$$\neg(\exists x_1 \cdots \exists x_m \ (Q''_1 \wedge \ldots \wedge Q''_m \Rightarrow P(t'_1, \ldots, t'_n))), \tag{A.3}$$

respectively

$$\neg(\exists x_1 \cdots \exists x_m \ (Q''_1 \wedge \ldots \wedge Q''_m \Rightarrow \mathscr{G}_F(t'_1, \ldots, t'_n))), \tag{A.4}$$

Proof. We proof the lemma by induction on the number of ambiguous subterms of φ. If there are no ambiguous subterms of $(t_1, \ldots, t_n)$, then the only cautious rewrite of φ is φ itself. Also, $t' = t$ for every subterm t of $(t_1, \ldots, t_n)$. It follows that (A.1) and (A.2) reduce to φ. This proves the base case for cautious rewrites of φ. The proof of the base case for $\neg\varphi$ is similar.
Now assume φ is of the form $P(t_1, \ldots, t_n)$ and has m ambiguous subterms, where $m > 0$. Any cautious rewrite of φ starts by a cautious one step rewrite, which transforms φ in a formula of the form $\forall x_i \ (f_i \neq x_i \vee \varphi')$, where φ' is obtained by replacing f_i in φ by x_i. Assume without loss of generality that the ambiguous terms $f_{i+1}, \ldots, f_m$ are all subterms of f_i, and $f_1, \ldots, f_{i-1}$ are subterms of φ'. By the induction hypothesis it follows that every cautious rewrite of φ' is equivalent to the formula

$$\forall x_1 \cdots \forall x_{i-1} \ (Q''_1 \wedge \ldots \wedge Q''_{i-1} \Rightarrow P(t'_1, \ldots, t'_n))$$

and every cautious rewrite of $f_i \neq x_i$ is equivalent to

$$\neg(\exists x_{i+1} \cdots \exists x_m \ (Q''_{i+1} \wedge \ldots \wedge Q''_m \wedge Q''_i)).$$

Therefore, every cautious rewrite of φ is equivalent to

$$\begin{aligned} \forall x_i \ (&\neg(\exists x_{i+1} \cdots \exists x_m \ (Q''_{i+1} \wedge \ldots \wedge Q''_m \wedge Q''_i)) \\ &\vee (\forall x_1 \cdots \forall x_{i-1} \ (Q''_1 \wedge \ldots \wedge Q''_{i-1} \Rightarrow P(t'_1, \ldots, t'_n)))), \end{aligned}$$

which is equivalent to (A.1). The proof of the inductive cases for $\neg P(t_1, \ldots, t_n)$, $F(t_1, \ldots, t_{n-1} = t_n$ and $F(t_1, \ldots, t_{n-1} \neq t_n$ are similar. ■

Proof of Proposition 3.1. By Lemma A.1, Proposition 3.1 follows by induction. Indeed, cautious rewrites only change the atomic subformulas, and Lemma A.1 states that all cautious rewrites of an atom are equivalent in FO(PF). ■

Proof of Proposition 3.2

The proof is by induction on the number of ambiguous subterms of φ. If φ has no ambiguous subterms, then the only rewrite of φ is φ itself. Hence in this case $\varphi_1 = \varphi_2 = \varphi$, and therefore $I\theta \models \mathscr{G}(\varphi_2)$ implies $I\theta \models \mathscr{G}(\varphi_1)$ for every structure I and variable assignment θ. This proves the base case.
For the inductive case, assume that φ has precisely n ambiguous subterms, where $n > 0$ and let φ_2 be a (possibly non-cautious) rewrite of φ. To obtain φ_2 from φ a sequence of one-step rewrites was applied. Let ψ_2 be the formula obtained after the first of these one-step rewrites. By induction it follows that if $I\theta \models \mathscr{G}(\varphi_2)$, then $I\theta \models \mathscr{G}(\chi)$, where χ is a cautious rewrite of ψ_2.
If ψ_2 was obtained by a cautious one-step rewrite on φ, χ is also a cautious rewrite of φ. This implies that χ is equivalent to φ_1 and therefore $I\theta \models \mathscr{G}(\varphi_2)$ implies $I\theta \models \mathscr{G}(\varphi_1)$.

If ψ_2 was obtained by a non-cautious one-step rewrite on φ, then some atomic subformula $P(\bar{t})$ of φ with ambiguous term t' was replaced by

$$\forall x\ (t' = x \Rightarrow P(\bar{t})[t'/x]) \tag{A.5}$$

if $P(\bar{t})$ occurs negatively in φ or by

$$\exists x\ (t' = x \wedge P(\bar{t})[t'/x]) \tag{A.6}$$

if it occurs positively. Assume $P(\bar{t})$ occurs negatively. The proof in the other case is similar. Consider the formula ψ_1 obtained from φ by replacing $P(\bar{t})$ by (A.6). Let I be an FO(PF) Σ-structure and θ a variable assignment. If $I\theta \models \exists x\ \mathscr{G}(t' = x)$, then $I\theta \models$ (A.5) iff $I\theta \models$ (A.6). If $I\theta \not\models \exists x\ \mathscr{G}(t' = x)$, then $I\theta \models$ (A.5) and $I\theta \not\models$ (A.6). Hence $I\theta \models$ (A.6) implies $I\theta \models$ (A.5). Since $P(\bar{t})$ occurs negatively in φ, it follows that for any I and θ, $I\theta \models \psi_2$ implies $I\theta \models \psi_1$. Because ψ_1 is obtained by a cautious rewrite, we obtain that $I\theta \models \mathscr{G}(\varphi_2)$ implies that $I\theta \models \mathscr{G}(\psi_2)$, which implies in turn that $I\theta \models \mathscr{G}(\varphi_1)$.

Proof of Proposition 4.2

Recall that we defined $|\tilde{I}|$ as the cardinality of the domain of $\tilde{I}$. We prove that every sequence $\tilde{I} = \tilde{J}_0 <_p \tilde{J}_1 <_p \ldots <_p \tilde{J}_n$ has length polynomial in $|\tilde{I}|$. Denote by N_P and N_F the number of predicate, respectively function, symbols in Σ. Let A_P and A_F be the maximum arity of a predicate, respectively function, symbol in Σ.
Since the sequence is increasing in precision, for every predicate symbol P the number of i such that $P^{\tilde{J}_i^{\mathrm{ct}}} \subsetneq P^{\tilde{J}_{i+1}^{\mathrm{ct}}}$ is at most $|\tilde{I}|^{A_P}$. Similarly, $P^{\tilde{J}_i^{\mathrm{cf}}}$ changes at most $|\tilde{I}|^{A_P}$ times in the sequence. By the same kind of reasoning it follows that the interpretation of each function symbol changes at most $2 \cdot |\tilde{I}|^{A_F+1}$ times in the sequence. Because $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$ is strictly increasing in precision, there is for every $0 \leq i < n$ at least one predicate P or function F such that $P^{\tilde{J}_i} \neq P^{\tilde{J}_{i+1}}$ or $F^{\tilde{J}_i} \neq F^{\tilde{J}_{i+1}}$. Combining these results gives a maximum length of

$$2 \cdot \left(|\tilde{I}|^{A_P} \cdot N_P + |\tilde{I}|^{A_F+1} \cdot N_F \right)$$

for the sequence $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$. Clearly, this is polynomial in $|\tilde{I}|$.

Proof of Proposition 4.10

If the technical details are omitted, the proof of Proposition 4.10 is straightforward. It consists of showing that each well-founded induction $\langle \tilde{K}_i \rangle_{0 \leq i \leq n}$ for $\Delta_{V,\tilde{I}}$ can be simulated by a $\mathscr{I}(V)$-refinement sequence $\langle \tilde{J}_i \rangle_{0 \leq i \leq n}$ and vice versa. The simulations depend on Proposition 2.7: a Σ-structure $\tilde{J}$ makes a formula φ true iff the corresponding Σ^{tf}-structure satisfies the formula φ^{ct}. Therefore, if in a $\mathscr{I}(V)$-refinement sequence, the INF propagator associated to $\forall \overline{x}\ (\varphi \Rightarrow P(\overline{x}))$

is used to obtain $\tilde{J}_{i+1}$ from $\tilde{J}_i$, the corresponding rule $\forall\overline{x}\ (P^{\mathrm{ct}}(\overline{x}) \leftarrow \varphi^{\mathrm{ct}})$ can be used to obtain $\tilde{K}_{i+1} = \tilde{J}_{i+1}^{\mathrm{tf}}$ from $\tilde{K}_i = \tilde{J}_i^{\mathrm{tf}}$, and vice versa. The simulation implies that the limit terminal well-founded induction for $\Delta_{V,\tilde{I}}$ corresponds to the limit of a terminal refinement sequence for $\mathscr{I}(V)$, which concludes the proof of the proposition.

We now give the proof with all technical details present. We prove Proposition 4.10 by showing that the two inequalities $\mathrm{wfm}_{\Delta_{V,\tilde{I}}}(\bot^{\leq_p}) \leq_p (\lim_{\mathscr{I}(V)}(\tilde{I}))^{\mathrm{tf}}$ and $\mathrm{wfm}_{\Delta_{V,\tilde{I}}}(\bot^{\leq_p}) \geq_p (\lim_{\mathscr{I}(V)}(\tilde{I}))^{\mathrm{tf}}$ hold. For any three-valued Σ^{tf}-structure $\tilde{K}$, the structure $\tilde{K}[A/\mathbf{f}]$ where A is the set of domain atoms that are unknown in $\tilde{K}$, is obviously two-valued. Hence there exists a unique Σ-structure, which we denote in this proof by $\Sigma(\tilde{K})$, such that $(\Sigma(\tilde{K}))^{\mathrm{tf}} = \tilde{K}[A/\mathbf{f}]$.

Lemma A.2. *Consider a well-founded induction $\langle\tilde{K}_i\rangle_{0\leq i\leq n}$ for $\Delta_{V,\tilde{I}}$ extending $\bot^{\leq_p}$ such that for each $i \geq 0$, there exists a domain atom $P_i(\overline{d}_i)$ such that $\tilde{K}_{i+1} = \tilde{K}_i[P_i(\overline{d}_i)/\mathbf{t}]$. Then there exists a sequence of Σ-structures $\langle\tilde{J}_i\rangle_{1\leq i\leq n}$ such that $\tilde{J}_0 = \tilde{I}$, $\Sigma(\tilde{K}_i) \leq_p \tilde{J}_i$ for every $0 \leq i \leq n$ and for every $0 \leq j < n$, either $\tilde{J}_{j+1} = \tilde{J}_j$ or $\tilde{J}_{j+1} = \mathscr{I}^{\varphi}(\tilde{J}_j)$ for some $\varphi \in V$.*

Proof. We prove the lemma by induction. The base case is trivial, since $\Sigma(\tilde{K}_0) = \bot^{\leq_p}$. For the inductive case, assume that the property is correct for sequences up to length $n-1$, i.e., there exists a sequence $\langle\tilde{J}_i\rangle_{1\leq i\leq n-1}$ with the desired property. We have to find a suitable structure $\tilde{J}_n$ that extends this sequence. Assume $\tilde{K}_n = \tilde{K}_{n-1}[P^{\mathrm{ct}}(\overline{d})/\mathbf{t}]$ for some predicate $P \in \Sigma_{\mathrm{pred}}$. Then there are two possibilities: either $P^{\tilde{I}}(\overline{d}) \geq_p \mathbf{t}$ or $P^{\tilde{I}}(\overline{d}) \not\geq_p \mathbf{t}$. In the former case, $P^{\tilde{J}_{n-1}} \geq_p \mathbf{t}$, and therefore $P^{\tilde{J}_{n-1}} \geq_p \tilde{K}_n$. It follows that it suffices to take $\tilde{J}_n = \tilde{J}_{n-1}$ in this case. In the other case, there exists a rule $\forall\overline{x}\ (P^{\mathrm{ct}}(\overline{x}) \leftarrow \psi^{\mathrm{ct}})$ in Δ_V such that $\tilde{K}_{n-1}[\overline{x}/\overline{d}](\psi) = \mathbf{t}$. Hence, V contains a sentence φ of the form $\forall\overline{x}\ (\psi \Rightarrow P(\overline{x}))$. By the induction hypothesis and Proposition 2.7, it follows that $\tilde{J}_{n-1}[\overline{x}/\overline{d}] \geq_p \psi$. As such, we can take $\tilde{J}_n = \mathscr{I}^{\varphi}(\tilde{J}_{n-1})$. The case where $\tilde{K}_n$ is of the form $\tilde{K}_{n-1}[P^{\mathrm{cf}}(\overline{d})/\mathbf{t}]$ is completely similar. ■

Lemma A.3. $\mathrm{wfm}_{\Delta_{V,\tilde{I}}}(\bot^{\leq_p}) \leq_p (\lim_{\mathscr{I}(V)}(\tilde{I}))^{\mathrm{tf}}$.

Proof. Because $\Delta_{V,\tilde{I}}$ is a positive definition, it follows that if $\langle\tilde{K}_i\rangle_{0\leq i\leq n}$ is a maximal well-founded induction such that every $\tilde{K}_{i+1}$ is of the form $\tilde{K}_i[P_i(\overline{d}_i)/\mathbf{t}]$, then $\Sigma(\tilde{K}_n) = \mathrm{wfm}_{\Delta_{V,\tilde{I}}}(\bot^{\leq_p})$. Lemma A.3 now follows from Lemma A.2. ■

Lemma A.4. $\mathrm{wfm}_{\Delta_{V,\tilde{I}}}(\bot^{\leq_p}) \geq_p (\lim_{\mathscr{I}(V)}(\tilde{I}))^{\mathrm{tf}}$.

Proof. Let $\langle\tilde{J}_i\rangle_{0\leq i\leq n}$ be a terminal refinement sequence from $\tilde{I}$. We show by induction on that for every i between 0 and n, there exists a well-founded induction for $\Delta_{V,\tilde{I}}$ extending $\bot^{\leq_p}$ with last element $\tilde{K}$ such that $\Sigma(\tilde{K}) \geq_p \tilde{J}_i$. Lemma A.4 then follows immediately.

For the base case, we have to prove that there exists a well-founded induction for $\Delta_{V,\tilde{I}}$ such that its last element is more precise than $\tilde{I}$. This follows from the fact that $\Sigma(\mathrm{wfm}_{\Delta_{\tilde{I}}}(\bot^{\leq_p})) = \tilde{I}$ and $\Delta_{\tilde{I}} \subseteq \Delta_{V,\tilde{I}}$.

For the inductive case, assume the above property is proven for $0 \leq i \leq n-1$, i.e., there exists a well-founded induction for $\Delta_{V,\tilde{I}}$ extending $\bot^{\leq_p}$ such that its last element is more precise than $\tilde{J}_{n-1}$. Denote such a well-founded induction by $\langle \tilde{K}_j \rangle_{0 \leq j \leq m-1}$. Let $\forall \overline{x}\ (\varphi \Rightarrow P(\overline{x}))$ be the sentence from V such that $\mathscr{I}^{\forall \overline{x}\ (\varphi \Rightarrow P(\overline{x}))}(\tilde{J}_{n-1}) = \tilde{J}_n$ (the proof is similar for a sentence of the form $\forall \overline{x}\ (\varphi \Rightarrow \neg P(\overline{x}))$). Denote the set of domain atoms $P^{\mathrm{ct}}(\overline{d})$ such that $P^{\tilde{J}_n}(\overline{d}) >_p P^{\tilde{J}_{n-1}}(\overline{d})$ by A. Let $\tilde{K}_m$ be the structure $\tilde{K}_{m-1}[A/\mathbf{t}]$. Because $\Sigma(\tilde{K}_{m-1}) \geq_p \tilde{J}_{n-1}$, it follows that $\Sigma(\tilde{K}_m) \geq_p \tilde{J}_n$. It remains to prove that $\langle \tilde{K}_j \rangle_{0 \leq j \leq m}$ is a well-founded induction for $\Delta_{V,\tilde{I}}$. Let $P^{\mathrm{ct}}(\overline{d})$ be a domain atom that is unknown in $\tilde{K}_{n-1}$ and true in $\tilde{K}_n$. Then $P^{\mathrm{ct}}(\overline{d})$ belongs to A, and therefore $\tilde{J}_{n-1}[\overline{x}/\overline{d}](\varphi) \geq_p \mathbf{t}$. It follows that $\tilde{K}_{m-1}[\overline{x}/\overline{d}] \models \varphi^{\mathrm{ct}}$. Since $\forall \overline{x}\ (P^{\mathrm{ct}}(\overline{x}) \leftarrow \varphi^{\mathrm{ct}})$ belongs to $\Delta_{V,\tilde{I}}$, $P^{\mathrm{ct}}(\overline{d})$ can indeed be made true in an extension of the well-founded induction $\langle \tilde{K}_j \rangle_{0 \leq j \leq m-1}$. ■

Proof of Theorem 4.12

The proof of Theorem 4.12 consists of two steps. First, we show that each ENF sentence φ of the form $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \varphi[\overline{x}])$ can be split into to the implications $\forall \overline{x}\ (P(\overline{x}) \Rightarrow \varphi[\overline{x}])$ and $\forall \overline{x}\ (P(\overline{x}) \Leftarrow \varphi[\overline{x}])$, denoted φ_L, respectively φ_R, without losing precision. That is, we prove that $\mathscr{O}^{\varphi} = \mathscr{L}_{\{\varphi_L, \varphi_R\}}$. This result is stated in Lemma A.5. Next, we show that there exists sets V_L and V_R of INF propagators such that $V_L \cup V_R = \mathrm{INF}(\varphi)$, $\lim_{\mathscr{I}(V_L) \cup \{\mathrm{INCO}\}} = \mathscr{O}^{\varphi_L}$ and $\lim_{\mathscr{I}(V_R) \cup \{\mathrm{INCO}\}} = \mathscr{O}^{\varphi_R}$. This is the content of Lemma A.14. Theorem 4.12 is a direct consequence of these lemmas.

Lemma A.5. *Let φ be the ENF sentence $\varphi = \forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \psi[\overline{x}])$ and φ_L, respectively φ_R, the sentences $\forall \overline{x}\ (P(\overline{x}) \Rightarrow \psi[\overline{x}])$, respectively $\forall \overline{x}\ (\psi[\overline{x}] \Rightarrow P(\overline{x}))$. Then $\mathscr{O}^{\varphi} = \mathscr{L}_{\{\varphi_L, \varphi_R\}}$.*

We present and prove several other lemmas in order to prove Lemma A.5. In each of the following lemmas, the notation is as introduced in Lemma A.5. Also, denote by T the theory $\{\varphi_L, \varphi_R\}$.

Lemma A.6. *Let $\tilde{I}$ be a structure such that $\mathscr{L}_T(\tilde{I}) = \tilde{I}$. If $\mathscr{O}^{\varphi}(\tilde{I}) \neq \tilde{I}$, then $\tilde{I}$ is strictly three-valued.*

Proof. If $\tilde{I}$ is two-valued, then $\tilde{I} \models \varphi_L$ and $\tilde{I} \models \varphi_R$. Hence $\tilde{I} \models \varphi$ and therefore $\mathscr{O}^{\varphi}(\tilde{I}) = \tilde{I}$. If $\tilde{I}$ is strictly four-valued, then $\tilde{I} = \top^{\leq_p}$ and $\mathscr{O}^{\varphi}(\tilde{I}) = \tilde{I}$. ■

Lemma A.7. *Let $\tilde{I}$ be a strictly three-valued structure such that $\mathscr{L}_T(\tilde{I}) = \tilde{I}$. Let $Q(\overline{d})$ be a domain atom such that $Q^{\tilde{I}}(\overline{d}) = \mathbf{u}$. If all literals in φ contain all variables of $\overline{x}$, then for every $\mathbf{v} \in \{\mathbf{t}, \mathbf{f}\}$ there exists a model M of φ such that $M \geq_p \tilde{I}$ and $M(Q(\overline{d})) = \mathbf{v}$.*

Proof. Let M_L and M_R be two-valued structures, more precise than $\tilde{I}$ such that $M_L \models \varphi_L$, $M_R \models \varphi_R$ and $Q^{M_L}(\overline{d}) = Q^{M_R}(\overline{d}) = \mathbf{v}$. Such structures exist

because $\tilde{I}$ is a fixpoint of $\mathscr{L}_T$. Let $\overline{d}_{x,1}$ and $\overline{d}_{x,2}$ be two different tuples of domain elements. Because no predicate occurs more than once in φ and all atoms in φ contain all variables of $\overline{x}$, it holds that no atom in $P(\overline{d}_{x,1}) \Leftrightarrow \psi[\overline{x}/\overline{d}_{x,1}]$ can be unified with an atom in $P(\overline{d}_{x,2}) \Leftrightarrow \psi[\overline{x}/\overline{d}_{x,2}]$ and vice versa. Therefore, the following construction of structure M is unambiguous. Denote by $\varphi_{\overline{d}_x}$ the sentence $P(\overline{d}_x) \Leftrightarrow \psi[\overline{x}/\overline{d}_x]$.

1. If predicate R does not occur in φ, let $R^M(\overline{d}') = R^{M_L}(\overline{d}')$ for every tuple of domain elements $\overline{d}'$;

2. If $M_L \models \varphi_{\overline{d}_x}$, define $R^M(\overline{d}') = R^{M_L}(\overline{d}')$ for every domain atom $R(\overline{d}')$ that unifies with an atom in $\varphi_{\overline{d}_x}$;

3. If $M_L \not\models \varphi_{\overline{d}_x}$ and $M_R \models \varphi_{\overline{d}_x}$, define $R^M(\overline{d}') = R^{M_R}(\overline{d}')$ for every domain atom $R(\overline{d}')$ that unifies with an atom in $\varphi_{\overline{d}_x}$;

4. If $M_L \not\models \varphi_{\overline{d}_x}$ and $M_R \not\models \varphi_{\overline{d}_x}$, define $P^M(\overline{d}_x)) = P^{M_R}(\overline{d}_x)$ and $R^M(\overline{d}') = R^{M_L}(\overline{d}')$ for all domain atoms that unify with an atom in $\psi[\overline{x}/\overline{d}_x]$.

Clearly, the resulting M is a two-valued structure and is more precise than $\tilde{I}$. Also, $Q^M(\overline{d}) = \mathbf{v}$. It remains to prove that $M \models \varphi$. Because of (2) and (3) in the construction of M, $M \models \varphi_{\overline{d}_x}$ if either $M_L \models \varphi_{\overline{d}_x}$ or $M_R \models \varphi_{\overline{d}_x}$. If neither $M_L \models \varphi_{\overline{d}_x}$ nor $M_R \models \varphi_{\overline{d}_x}$, then $M_L \not\models P(\overline{d}_x) \Leftarrow \psi[\overline{x}/\overline{d}_x]$ and $M_R \not\models P(\overline{d}_x) \Rightarrow \psi[\overline{x}/\overline{d}_x]$. Hence $M_L \models \psi[\overline{x}/\overline{d}_x]$ and $M_R \models P(\overline{d}_x)$. Because of step (4) in the construction of M, we conclude that also in this case $M \models \varphi_{\overline{d}_x}$. It follows that $M \models \varphi$. ■

Lemma A.8. *Let $\tilde{I}$ be a strictly three-valued structure such that $\mathscr{L}_T(\tilde{I}) = \tilde{I}$. Let $Q(\overline{d})$ be a domain atom such that $Q^{\tilde{I}}(\overline{d}) = \mathbf{u}$. If ψ is of the form $L_1(\overline{x}_1) \wedge L_2(\overline{x}_2)$ or $L_1(\overline{x}_1) \vee L_2(\overline{x}_2)$, then for every $\mathbf{v} \in \{\mathbf{t}, \mathbf{f}\}$ there exists a model M of φ such that $M \geq_p \tilde{I}$ and $Q^M(\overline{d}) = \mathbf{v}$.*

Proof. We prove the case where φ is of the form $\forall\overline{z}_1\forall\overline{z}_2\forall\overline{z}_3\ (P[\overline{z}_1, \overline{z}_2, \overline{z}_3] \Leftrightarrow R[\overline{z}_1, \overline{z}_2] \wedge S[\overline{z}_1, \overline{z}_3])$. The proofs for the other cases are similar.
If $\mathbf{v} = \mathbf{f}$, let V be the set of all domain atoms that are unknown in $\tilde{I}$ and define $M = \tilde{I}[V/\mathbf{f}]$. Clearly, $Q^M(\overline{d})) = \mathbf{f}$ and $M \geq_p \tilde{I}$. It remains to prove that $M \models \varphi$. We do so by showing that for all tuples $\overline{d}_1, \overline{d}_2, \overline{d}_3$ of domain elements, $M \models P(\overline{d}_1, \overline{d}_2, \overline{d}3)$ iff $M \models R(\overline{d}_1, \overline{d}_2) \wedge S(\overline{d}_1, \overline{d}_3)$.

- If $M \models P(\overline{d}_1, \overline{d}_2, \overline{d}_3)$, then $P^{\tilde{I}}(\overline{d}_1, \overline{d}_2, \overline{d}_3) = \mathbf{t}$ and therefore $R^{\tilde{I}}(\overline{d}_1, \overline{d}_2) = S^{\tilde{I}}(\overline{d}_1, \overline{d}_3) = \mathbf{t}$. As such, $M \models R(\overline{d}_1, \overline{d}_2) \wedge S(\overline{d}_1, \overline{d}_3)$.

- Vice versa, if $M \models R(\overline{d}_1, \overline{d}_2) \wedge S(\overline{d}_1, \overline{d}_3)$, then $R^{\tilde{I}}(\overline{d}_1, \overline{d}_2) = S^{\tilde{I}}(\overline{d}_1, \overline{d}_3) = \mathbf{t}$. It follows that $P^{\tilde{I}}(\overline{d}_1, \overline{d}_2, \overline{d}_3) = \mathbf{t}$ and hence $M \models P(\overline{d}_1, \overline{d}_2, \overline{d}_3)$.

We conclude that $M \models \varphi$.
Now let $\mathbf{v} = \mathbf{t}$. Denote by respectively ι_1, ι_2 and ι_3 the three INF sentences

$$\forall \overline{z}_1 \forall \overline{z}_2 \ ((\exists \overline{z}_3 \ P[\overline{z}_1, \overline{z}_2, \overline{z}_3]) \Rightarrow R(\overline{z}_1, \overline{z}_2)),$$
$$\forall \overline{z}_1 \forall \overline{z}_3 \ ((\exists \overline{z}_2 \ P[\overline{z}_1, \overline{z}_2, \overline{z}_3]) \Rightarrow S(\overline{z}_1, \overline{z}_3)),$$
$$\forall \overline{z}_1 \forall \overline{z}_2 \forall \overline{z}_3 \ (R(\overline{z}_1, \overline{z}_2) \wedge S(\overline{z}_1, \overline{z}_3) \Rightarrow P[\overline{z}_1, \overline{z}_2, \overline{z}_3]).$$

Let $\tilde{J}$ be the structure $\mathscr{I}^{\iota_3}(\mathscr{I}^{\iota_2}(\mathscr{I}^{\iota_1}(\tilde{I}[Q(\overline{d})/\mathbf{t}])))$, V the set of domain atoms that are unknown in $\tilde{J}$ and M the structure $\tilde{J}[V/\mathbf{f}]$. If $\tilde{J}$ is three-valued then by similar reasoning as above we obtain that $Q^M(\overline{d}) = \mathbf{t}$, $M \geq_p \tilde{I}$ and $M \models \varphi$. Hence it suffices to show that $\tilde{J}$ is indeed three-valued. Denote $\tilde{J}_0 = \tilde{I}[Q(\overline{d})/\mathbf{t}]$, $\tilde{J}_1 = \mathscr{I}^{\iota_1}(\tilde{J}_0)$ and $\tilde{J}_2 = \mathscr{I}^{\iota_2}(\tilde{J}_1)$. If $\tilde{J}_1 \neq \tilde{J}_0$, then $Q(\overline{d}) = P(\overline{d}_1, \overline{d}_2, \overline{d}_3)$ for some tuples of domain elements $\overline{d}_1, \overline{d}_2, \overline{d}_3$. Because $\mathscr{O}^{\varphi_L}(\tilde{I}) = \tilde{I}$ and $Q^{\tilde{I}}(\overline{d}) = \mathbf{u}$, it follows that $R^{\tilde{I}}(\overline{d}_1, \overline{d}_2) \leq_p \mathbf{t}$ and therefore $R^{\tilde{J}_1}(\overline{d}_1, \overline{d}_2) = \mathbf{t}$. Hence $\tilde{J}_1$ is three-valued. Similarly, $\tilde{J}_2$ is three-valued. Now assume that $\tilde{J}$ is strictly four-valued. Then $P^{\tilde{I}}(\overline{d}_1, \overline{d}_2, \overline{d}_3) = \mathbf{f}$ for some tuples $\overline{d}_1, \overline{d}_2, \overline{d}_3$ and $R^{\tilde{J}_2}(\overline{d}_1, \overline{d}_2) = S^{\tilde{J}_2}(\overline{d}_1, \overline{d}_3) = \mathbf{t}$. Since $\mathscr{O}^{\varphi_R}(\tilde{I}) = \tilde{I}$, we have that $R^{\tilde{I}}(\overline{d}_1, \overline{d}_2) = S^{\tilde{I}}(\overline{d}_1, \overline{d}_3) = \mathbf{u}$. Therefore one of the following is true:

- $Q(\overline{d}) = R(\overline{d}_1, \overline{d}_2) = S(\overline{d}_1, \overline{d}_3)$
- $Q(\overline{d}) = S(\overline{d}_1, \overline{d}_3) = P(\overline{d}_1, \overline{d}_2, \overline{d}_4)$ and $\overline{d}_3 \neq \overline{d}_4$
- $Q(\overline{d}) = R(\overline{d}_1, \overline{d}_2) = P(\overline{d}_1, \overline{d}_5, \overline{d}_3)$ and $\overline{d}_2 \neq \overline{d}_5$
- $Q(\overline{d}) = P(\overline{d}_1, \overline{d}_2, \overline{d}_4) = P(\overline{d}_1, \overline{d}_5, \overline{d}_3)$, $\overline{d}_2 \neq \overline{d}_5$ and $\overline{d}_3 \neq \overline{d}_4$

Because φ is an ENF sentence, it does not contain a predicate more than once. As such, none of the four statements can be true. This contradiction proves that $\tilde{J}$ is three-valued. ■

Lemma A.9. *Let $\tilde{I}$ be a strictly three-valued structure such that $\mathscr{L}_T(\tilde{I}) = \tilde{I}$. Let $Q(\overline{d})$ be a domain atom such that $Q^{\tilde{I}}(\overline{d}) = \mathbf{u}$. Then for every $\mathbf{v} \in \{\mathbf{t}, \mathbf{f}\}$ there exists a model M of φ such that $M \geq_p \tilde{I}$ and $M(Q(\overline{d})) = \mathbf{v}$.*

Proof. Follows directly from Lemma A.7 and Lemma A.8 since these lemmas cover all ENF sentences. ■

Proof of Lemma A.5. Let $\tilde{I}$ be a three-valued structure and denote $\tilde{J} = \mathscr{O}^{\varphi}(\tilde{I})$ and $\tilde{K} = \mathscr{L}_T(\tilde{I})$. If $\tilde{K}$ is not strictly three-valued, then by Lemma A.6, $\tilde{J} = \tilde{K}$. If $\tilde{K}$ is strictly three-valued, then for any domain atom $Q(\overline{d})$ such that $Q^{\tilde{K}}(\overline{d}) = \mathbf{u}$, Lemma A.9 implies that also $Q^{\tilde{J}}(\overline{d}) = \mathbf{u}$. We conclude that also in this case, $\tilde{J} = \tilde{K}$. ■

ENF sentences of the form $\forall \overline{x} \ (P(\overline{x}) \Leftrightarrow L_1 \wedge L_2)$ and $\forall \overline{x} \ (P(\overline{x}) \Leftrightarrow L_1 \vee L_2)$ can be split further into three implications, as indicated by the following lemma. A minor adaptation in the proof of Lemma A.8 suffices to prove it.

φ	IMPL(φ)
$\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow L[\overline{x}])$	$\forall\overline{x}\ (P(\overline{x}) \Rightarrow L[\overline{x}])$ $\forall\overline{x}\ (L[\overline{x}] \Rightarrow P(\overline{x}))$
$\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow \forall y\ L[\overline{x}, y])$	$\forall\overline{x}\forall y\ (P(\overline{x}) \Rightarrow L[\overline{x}, y])$ $\forall\overline{x}\ ((\forall y\ L[\overline{x}, y]) \Rightarrow P(\overline{x}))$
$\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow \exists y\ L[\overline{x}, y])$	$\forall\overline{x}\ (P(\overline{x}) \Rightarrow (\exists y\ L[\overline{x}, y]))$ $\forall\overline{x}\ ((\exists y\ L[\overline{x}, y]) \Rightarrow P(\overline{x}))$
$\forall\overline{x}\forall\overline{y}\forall\overline{z}\ (P(\overline{x}, \overline{y}, \overline{z})$ $\Leftrightarrow L_1[\overline{x}, \overline{y}] \wedge L_2[\overline{x}, \overline{z}])$	$\forall\overline{x}\forall\overline{y}((\exists\overline{z}\ P(\overline{x}, \overline{y}, \overline{z})) \Rightarrow L_1[\overline{x}, \overline{y}])$ $\forall\overline{x}\forall\overline{z}((\exists\overline{y}\ P(\overline{x}, \overline{y}, \overline{z})) \Rightarrow L_2[\overline{x}, \overline{z}])$ $\forall\overline{x}\forall\overline{y}\forall\overline{z}\ (L_1[\overline{x}, \overline{y}] \wedge L_2[\overline{x}, \overline{z}] \Rightarrow P(\overline{x}, \overline{y}, \overline{z}))$
$\forall\overline{x}\forall\overline{y}\forall\overline{z}\ (P(\overline{x}, \overline{y}, \overline{z})$ $\Leftrightarrow L_1[\overline{x}, \overline{y}] \vee L_2[\overline{x}, \overline{z}])$	$\forall\overline{x}\forall\overline{y}\forall\overline{z}\ (P(\overline{x}, \overline{y}, \overline{z}) \Rightarrow L_1[\overline{x}, \overline{y}] \vee L_2[\overline{x}, \overline{z}])$ $\forall\overline{x}\forall\overline{y}\forall\overline{z}\ (L_1[\overline{x}, \overline{y}] \Rightarrow P(\overline{x}, \overline{y}, \overline{z}))$ $\forall\overline{x}\forall\overline{y}\forall\overline{z}\ (L_2[\overline{x}, \overline{z}] \Rightarrow P(\overline{x}, \overline{y}, \overline{z}))$

Table A.1: Splitting ENF sentences into implications

Lemma A.10. *If φ is the ENF sentence $\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow L_1 \wedge L_2)$, then*

$$\mathscr{O}^{\varphi} = \mathscr{L}_{\{\varphi_R, \forall\overline{x}\ (P(\overline{x}) \Rightarrow L_1), \forall\overline{x}\ (P(\overline{x}) \Rightarrow L_2)\}}.$$

If φ is the ENF sentence $\forall\overline{x}\ (P(\overline{x}) \Leftrightarrow L_1 \vee L_2)$, then

$$\mathscr{O}^{\varphi} = \mathscr{L}_{\{\varphi_L, \forall\overline{x}\ (P(\overline{x}) \Leftarrow L_1), \forall\overline{x}\ (P(\overline{x}) \Leftarrow L_2)\}}.$$

Lemma A.5 and Lemma A.10 show how to split an ENF sentence into different implications while preserving the precision of the propagators. For an ENF sentence φ we denote this set of implications by IMPL(φ). See Table A.1 for an overview. Using this notation, the following lemma summarizes Lemma A.5 and Lemma A.10.

Lemma A.11. *$\mathscr{O}^{\varphi}(\tilde{I}) = \mathscr{L}_{\mathrm{IMPL}(\varphi)}(\tilde{I})$ for every ENF sentence φ and structure $\tilde{I}$.*

Observe that for every ENF sentence φ and $\psi \in \mathrm{IMPL}(\varphi)$, ψ can be rewritten to an equivalent formula that has one of the following forms

$$\forall\overline{x}\ (L_1[\overline{x}] \Rightarrow L_2[\overline{x}]), \tag{A.7}$$
$$\forall\overline{x}\ ((\exists y\ L_1[\overline{x}, y]) \Rightarrow L_2[\overline{x}]), \tag{A.8}$$
$$\forall\overline{x}\ ((\forall y\ L_1[\overline{x}, y]) \Rightarrow L_2[\overline{x}]), \tag{A.9}$$
$$\forall\overline{x}\forall\overline{y}\forall\overline{z}\ (L_1[\overline{x}, \overline{y}] \wedge L_2[\overline{x}, \overline{z}] \Rightarrow L_3[\overline{x}, \overline{y}, \overline{z}]), \tag{A.10}$$

where L_1, L_2 and L_3 are literals over different predicates. For each formula ψ of the form (A.7), (A.8), (A.9) or (A.10), there exists a set II(ψ) of INF sentences such that

ψ	$\mathrm{II}(\psi)$
$\forall\overline{x}\ (L_1[\overline{x}] \Rightarrow L_2[\overline{x}])$	$\forall\overline{x}\ (L_1[\overline{x}] \Rightarrow L_2[\overline{x}])$ $\forall\overline{x}\ (\neg L_2[\overline{x}] \Rightarrow \neg L_1[\overline{x}])$
$\forall\overline{x}\ ((\exists y\ L_1[\overline{x}, y]) \Rightarrow L_2[\overline{x}])$	$\forall\overline{x}\ ((\exists y\ L_1[\overline{x}, y]) \Rightarrow L_2[\overline{x}])$ $\forall\overline{x}\forall y\ (\neg L_2[\overline{x}] \Rightarrow \neg L_1[\overline{x}, y])$
$\forall\overline{x}\ ((\forall y\ L_1[\overline{x}, y]) \Rightarrow L_2[\overline{x}])$	$\forall\overline{x}\ ((\forall y\ L_1[\overline{x}, y]) \Rightarrow L_2[\overline{x}])$ $\forall\overline{x}\forall y\ ((\neg L_2(\overline{x}) \wedge \forall y'\ (y \neq y' \Rightarrow L_1[\overline{x}, y']))$ $\Rightarrow \neg L_1[\overline{x}, y])$
$\forall\overline{x}\forall\overline{y}\forall\overline{z}\ (L_1[\overline{x}, \overline{y}] \wedge L_2[\overline{x}, \overline{z}]$ $\Rightarrow L_3[\overline{x}, \overline{y}, \overline{z}])$	$\forall\overline{x}\forall\overline{y}\forall\overline{z}\ (L_1[\overline{x}, \overline{y}] \wedge L_2[\overline{x}, \overline{z}] \Rightarrow L_3[\overline{x}, \overline{y}, \overline{z}])$ $\forall\overline{x}\forall\overline{y}((\exists\overline{z}\ (L_2[\overline{x}, \overline{z}] \wedge \neg L_3[\overline{x}, \overline{y}, \overline{z}])) \Rightarrow \neg L_1[\overline{x}, \overline{y}])$ $\forall\overline{x}\forall\overline{z}((\exists\overline{y}\ (L_1[\overline{x}, \overline{z}] \wedge \neg L_3[\overline{x}, \overline{y}, \overline{z}])) \Rightarrow \neg L_2[\overline{x}, \overline{z}])$

Table A.2: From implications to INF sentences

- Each sentence of $\mathrm{II}(\psi)$ is equivalent to ψ
- If P occurs positively in ψ, then $\mathrm{II}(\psi)$ contains a sentence of the form $\forall\overline{x}\ (\chi \Rightarrow P(\overline{x}))$.
- If P occurs negatively in ψ, then $\mathrm{II}(\psi)$ contains a sentence of the form $\forall\overline{x}\ (\chi \Rightarrow \neg P(\overline{x}))$.

All these sets are shown in Table A.2. Using Table 4.1, Table A.1 and Table A.2, it is straightforward to check the correctness of the following lemma.

Lemma A.12. $\mathrm{II}(\mathrm{IMPL}(\varphi)) = \mathrm{INF}(\varphi)$ *for every ENF sentence* φ.

To prove Theorem 4.12, it now suffices to show that $\mathscr{O}^{\psi} = \textsc{inco} \circ \lim_{\mathrm{INF}(\mathrm{II}(\psi))}$ for every sentence ψ of the form (A.7), (A.8), (A.9) or (A.10). We split the proof over two lemmas. The first one covers the cases where ψ is of the form, (A.7), (A.8) or (A.10), the second one covers the remaining case.

Lemma A.13. *Let* φ *be a sentence such that no predicate occurs twice in* φ*, and let* $P_1, \ldots, P_n$ *be the predicates that occur in* φ*. Let* $\varphi_1, \ldots, \varphi_n$ *be INF sentences of the form* $\forall\overline{x}\ (\psi_i \Rightarrow P_i(\overline{x}))$ *or* $\forall\overline{x}\ (\psi_i \Rightarrow \neg P_i(\overline{x}))$*. If for each* $1 \leq i \leq n$*,* φ_i *is equivalent to* φ *and no predicate occurs twice in* φ_i*, then* $\mathscr{O}^{\varphi} = \textsc{inco} \circ \lim_{\{\mathscr{I}^{\varphi_1}, \ldots, \mathscr{I}^{\varphi_n}\}}$.

Proof. Denote the set $\{\mathscr{I}^{\varphi_1}, \ldots, \mathscr{I}^{\varphi_n}\}$ by V and let $\tilde{I}$ be a fixpoint of $\lim_V$. We have to prove that $\mathscr{O}^{\varphi}(\tilde{I}) = \textsc{inco}(\tilde{I})$. If $\tilde{I}$ is strictly four-valued, then both $\textsc{inco}(\tilde{I})$ and $\mathscr{O}^{\varphi}$ are equal to $\top^{\leq_p}$. If on the other hand, $\tilde{I}$ is two-valued, then $\tilde{I} \models \varphi_1$. Because φ_1 is equivalent to φ, also $\tilde{I} \models \varphi$ and therefore $\mathscr{O}^{\varphi}(\tilde{I}) = \tilde{I}$ also holds in this case. Finally, assume $\tilde{I}$ is strictly three-valued, and let $Q(\overline{d})$ be a domain atom such that $Q^{\tilde{I}}(\overline{d}) = \mathbf{u}$. It suffices to show that for every $\mathbf{v} \in \{\mathbf{t}, \mathbf{f}\}$ there exists a model M of φ such that $M \geq_p \tilde{I}$ and $Q^M(\overline{d}) = \mathbf{v}$.

Without loss of generality, assume that φ_1 is of the form $\forall\overline{x}\ (\psi \Rightarrow L[\overline{x}])$, where $L[\overline{x}]$ is either the literal $Q(\overline{x})$ or $\neg Q(\overline{x})$. Denote the structure $\tilde{I}[Q(\overline{d})/\mathbf{v}]$ by $\tilde{J}$. If $\tilde{J}[\overline{x}/\overline{d}](L[\overline{x}]) = \mathbf{t}$, then $\tilde{J}(\varphi_1) \geq_t \tilde{I}(\varphi_1)$ and since $\mathscr{I}^{\varphi_1}(\tilde{I}) = \tilde{I}$, it follows that $\tilde{J}(\varphi_1) \geq_t \mathbf{u}$. Because φ_1 contains no predicate twice, it follows from Lemma 2.5 that there exists a model $M \geq_p \tilde{J}$ such that $M \models \varphi_1$. Hence $M \models \varphi$, $M \geq_p \tilde{I}$ and $Q^M(\overline{d}) = \mathbf{v}$, as desired.

Now consider the case where $\tilde{J}[\overline{x}/\overline{d}](L[\overline{x}]) = \mathbf{f}$. Since $\tilde{I}$ is a fixpoint of $\lim_V$, $\tilde{I}[\overline{x}/\overline{d}](L[\overline{x}]) = \mathbf{u}$ and Q does not occur in ψ, it follows that $\tilde{I}[\overline{x}/\overline{d}](\psi) \leq_t \mathbf{u}$. Therefore also $\tilde{J}[\overline{x}/\overline{d}](\psi) \leq_t \mathbf{u}$. Because no predicate occurs twice in φ_1, Lemma 2.5 guarantees we can construct a structure $M \geq_p \tilde{J}$ such that for every tuple $\overline{d}'$, $M[\overline{x}/\overline{d}'] \models \psi$ iff $\tilde{J}[\overline{x}/\overline{d}'](\psi) = \mathbf{t}$ and $M[\overline{x}/\overline{d}'] \models L[\overline{x}]$ iff $\tilde{J}[\overline{x}/\overline{d}'](L[\overline{x}]) \geq_t \mathbf{u}$. It is straightforward to verify that M is a model of φ_1, and therefore of φ. ■

Lemma A.14. *Let φ be the sentence $\forall\overline{x}\ ((\forall y\ L_1[\overline{x}, y]) \Rightarrow L_2[\overline{x}])$ and ψ the sentence $\forall\overline{x}\forall y\ ((\neg L_2(\overline{x}) \wedge \forall y'\ (y \neq y' \Rightarrow L_1[\overline{x}, y'])) \Rightarrow \neg L_1[\overline{x}, y])$. Then $\mathscr{O}^{\varphi} = \text{INCO} \circ \lim_{\{\mathscr{I}^{\varphi}, \mathscr{I}^{\psi}\}}$.*

Proof. Similarly as in the proof of Lemma A.13, it suffices to show that for any three-valued fixpoint $\tilde{I}$ of $\lim_{\{\mathscr{I}^{\varphi}, \mathscr{I}^{\psi}\}}$, truth value $\mathbf{v} \in \{\mathbf{t}, \mathbf{f}\}$ and domain atom $Q(\overline{d})$ such that $Q^{\tilde{I}}(\overline{d}) = \mathbf{u}$, there exists a structure $M \geq_p \tilde{I}$ such that $M \models \varphi$ and $Q^M(\overline{d}) = \mathbf{v}$. We assume that $L_1[\overline{x}, y]$ is the positive literal $R(\overline{x}, y)$ and $L_2[\overline{x}]$ is the positive literal $P(\overline{x})$. The proof is similar if $L_1[\overline{x}, y]$ and/or $L_2[\overline{x}]$ are negative literals.

The case where $Q = P$ can be proven in a similar way as Lemma A.13, as can be the case where $Q = R$ and $\mathbf{v} = \mathbf{f}$. Now let $Q(\overline{d}) = R(\overline{d}_x, d_y)$ and $\mathbf{v} = \mathbf{t}$ and denote $\tilde{J} = \tilde{I}[R(\overline{d}_x, d_y)/\mathbf{t}]$. If $\tilde{J}(\varphi) \neq \mathbf{f}$, then there exists a structure $M \geq_p \tilde{J}$ such that $M \models \varphi$. Towards a contradiction, assume $\tilde{J}(\varphi) = \mathbf{f}$. Since $\tilde{I}$ is a three-valued fixpoint of $\mathscr{I}^{\varphi}$, $\tilde{I}(\varphi) \neq \mathbf{f}$. Therefore $\tilde{J}[\overline{x}/\overline{d}_x](\forall y\ R(\overline{x}, y)) = \mathbf{t}$ and $P^{\tilde{J}}(\overline{d}_x) = \mathbf{f}$. It follows that $P^{\tilde{I}}(\overline{d}_x) = \mathbf{f}$ and $R^{\tilde{I}}(\overline{d}_x, d'_y) = \mathbf{t}$ for each $d'_y \neq d_y$. Thus $\tilde{I}[\overline{x}/\overline{d}_x][y/d_y](\neg P(\overline{x}) \wedge \forall y'\ (y \neq y' \Rightarrow R(\overline{x}, y'])))) = \mathbf{t}$. Because $\tilde{I}$ is a fixpoint of $\mathscr{I}^{\psi}$, we derive that $R^{\tilde{I}}(\overline{d}_x, d_y) = \mathbf{f}$, which contradicts the fact that $R^{\tilde{I}}(\overline{d}_x, d_y) = \mathbf{u}$. ■

The following lemma is a corollary of Lemma A.13 and Lemma A.14.

Lemma A.15. *$\mathscr{O}^{\varphi} = \text{INCO} \circ \lim_{\text{INF}(\text{II}(\varphi))}$ for each φ of the form* (A.7), (A.8), (A.9) *or* (A.10).

Theorem 4.12 is a corollary of Lemma A.11, Lemma A.12 and Lemma A.15.

Proof of Proposition 4.17

For $T = \{(4.25), (4.26)\}$, the theory INF(T) contains, a.o., the sentences

$$\forall \overline{x}\ (\top \Rightarrow Aux_1(\overline{x})) \tag{A.11}$$
$$\forall \overline{x}\ \forall y\ ((Aux_1(\overline{x}) \wedge \forall y'\ (y \neq y' \Rightarrow \neg \mathscr{G}_F(\overline{x}, y'))) \Rightarrow \mathscr{G}_F(\overline{x}, y)) \tag{A.12}$$
$$\forall \overline{x} \forall y_1 \forall y_2\ (\top \Rightarrow Aux_2(\overline{x}, y_1, y_2)) \tag{A.13}$$
$$\forall \overline{x} \forall y_2 \forall y_2\ (Aux_2(\overline{x}, y_1, y_2) \wedge y_1 \neq y_2 \Rightarrow Aux_3(\overline{x}, y_1, y_2)) \tag{A.14}$$
$$\forall \overline{x} \forall y_1\ ((\exists y_2\ (Aux_3(\overline{x}, y_1, y_2) \wedge \mathscr{G}_F(\overline{x}, y_2))) \Rightarrow \mathscr{G}_F(\overline{x}, y_1)) \tag{A.15}$$

Let $\tilde{J}$ be a fixpoint of $\lim_{\mathscr{I}(\mathrm{INF}(T))}$. Because (A.11) and (A.13) belong to INF(T), $Aux_1^{\tilde{J}}(\overline{d}) \geq_p \mathbf{t}$ and $Aux_2^{\tilde{J}}(\overline{d}, d_1, d_2) \geq_p \mathbf{t}$ for every $\overline{d}$, d_1 and d_2. Since formula (A.14) belongs to INF(T), it the follows that $Aux_3^{\tilde{J}}(\overline{d}, d_1, d_2) \geq_p \mathbf{t}$ for every tuple $\overline{d}$ and domain elements $d_1 \neq d_2$. Because (A.12) is part of INF(T), it follows that $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d_y) \geq_p \mathbf{t}$ if $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d'_y) \geq_p \mathbf{f}$ for every $d'_y \neq d_y$. Finally, if $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d_y) \geq_p \mathbf{t}$, then $\mathscr{G}_F^{\tilde{J}}(\overline{d}, d'_y) \geq_p \mathbf{f}$ for every $d'_y \neq d_y$, since (A.15) belongs to INF(T). It is now straightforward to check that $\tilde{J}$ satisfies the four conditions from Proposition 2.6.

Proof of Proposition 4.22

We prove the case where φ is the formula $\textsc{card}\{\overline{x} \mid \psi\} \leq y$. The other cases can be proven in a similar way.
Let φ be the formula $\textsc{card}\{\overline{x} \mid \psi\} \leq y$. Then φ^{ct} and φ^{cf} are, respectively, the formulas

$$\exists y_1 \exists y_2\ (\textsc{card}\{\overline{x} \mid \psi^{\mathrm{cf}}\} \geq y_1 \wedge \textsc{card}\{\overline{x} \mid \top\} \leq y_2 \wedge y_2 - y_1 \leq y) \tag{A.16}$$

and

$$\exists z\ (\textsc{card}\{\overline{x} \mid \psi^{\mathrm{ct}}\} \geq z \wedge z > y). \tag{A.17}$$

We have to show that for every three-valued structure $\tilde{I}$ and variable assignment θ,

1. $\tilde{I}^{\mathrm{tf}}\theta$ does not satisfy both (A.16) and (A.17);
2. $\tilde{I}\theta(\varphi) = \mathbf{t}$ iff $\tilde{I}^{\mathrm{tf}}\theta \models$ (A.16);
3. $\tilde{I}\theta(\varphi) = \mathbf{f}$ iff $\tilde{I}^{\mathrm{tf}}\theta \models$ (A.17).

To prove the first statement, observe that because of Proposition 2.7, $\tilde{I}^{\mathrm{tf}}\theta$ does not satisfy both ψ^{ct} and ψ^{cf}, and therefore

$$\tilde{I}^{\mathrm{tf}}\theta \models \textsc{card}\{\overline{x} \mid \psi^{\mathrm{ct}}\} + \textsc{card}\{\overline{x} \mid \psi^{\mathrm{cf}}\} + \textsc{card}\{\overline{x} \mid \neg\psi^{\mathrm{ct}} \wedge \neg\psi^{\mathrm{cf}}\} = \textsc{card}\{\overline{x} \mid \top\}. \tag{A.18}$$

If $\tilde{I}^{\mathrm{tf}}\theta \models$ (A.17), then $\tilde{I}^{\mathrm{tf}}\theta(\textsc{card}\{\overline{x} \mid \psi^{\mathrm{ct}}\}) > \theta(y)$ and hence

$$\tilde{I}^{\mathrm{tf}}\theta \models y + \textsc{card}\{\overline{x} \mid \psi^{\mathrm{cf}}\} + \textsc{card}\{\overline{x} \mid \neg\psi^{\mathrm{ct}} \wedge \neg\psi^{\mathrm{cf}}\} < \textsc{card}\{\overline{x} \mid \top\}.$$

From the fact that $\tilde{I}^{\mathrm{tf}}\theta(\textsc{card}\{\overline{x} \mid \neg\psi^{\mathrm{ct}} \wedge \neg\psi^{\mathrm{cf}}\}) \geq 0$, it follows that

$$y < \textsc{card}\{\overline{x} \mid \top\} - \textsc{card}\{\overline{x} \mid \psi^{\mathrm{cf}}\}.$$

Therefore $\tilde{I}^{\mathrm{tf}}\theta \not\models$ (A.16) if $\tilde{I}^{\mathrm{tf}}\theta \models$ (A.17), which concludes the proof of the first statement.

To prove the second statement, note that for a three-valued set $\tilde{V}$, the maximal value of $\{\textsc{card}(v) \mid v \sqsubset \tilde{V}\}$ is given by $\textsc{card}\{\overline{d}^{\mathbf{v}} \in \tilde{V} \mid \mathbf{v} \neq \mathbf{f}\}$. From Definition 3.15, it now follows that $\tilde{I}\theta(\varphi) = \mathbf{t}$ iff $\tilde{I}\theta(\textsc{card}\{\overline{x} \mid \psi\})_{\max} \leq \theta(y)$ iff $\tilde{I}^{\mathrm{tf}}\theta(\textsc{card}\{\overline{x} \mid \psi^{\mathrm{ct}}\}) + \tilde{I}^{\mathrm{tf}}\theta(\textsc{card}\{\overline{x} \mid \neg\psi^{\mathrm{ct}} \wedge \neg\psi^{\mathrm{cf}}\}) \leq \theta(y)$. By (A.18) we then have that $\tilde{I}\theta(\varphi) = \mathbf{t}$ iff $\tilde{I}^{\mathrm{tf}}\theta(\textsc{card}\{\overline{x} \mid \top\} - \textsc{card}\{\overline{x} \mid \psi^{\mathrm{cf}}\}) \leq \theta(y)$ iff $\tilde{I}^{\mathrm{tf}}\theta \models$ (A.16).

The proof of the third statement is similar to the proof of the second statement.

Aggregates with negative weights

In this section, we extend the definitions of φ^{ct} and φ^{cf} to formulas φ that may contain aggregates of the form $\textsc{f}(\{(n, \overline{x}) \mid \psi\})$ where $\mathfrak{s}(n)$ may contain negative numbers and $\textsc{f} \neq \textsc{prod}$. As such, we generalize Proposition 4.22 to arbitrary finite structures and theories that do not contain PROD.

Sum aggregate

Recall that the aim is to find for every formula φ two monotone formulas φ^{ct} and φ^{cf} over Σ^{tf} such that for any finite three-valued structure $\tilde{I}$, $\tilde{I}(\varphi) = \tilde{I}^{\mathrm{tf}}(\varphi^{\mathrm{ct}}, \varphi^{\mathrm{cf}})$. Equivalently, $\tilde{I}^{\mathrm{tf}} \models \varphi^{\mathrm{ct}}$ iff $\tilde{I}(\varphi) = \mathbf{t}$ and $\tilde{I}^{\mathrm{tf}} \models \varphi^{\mathrm{cf}}$ iff $\tilde{I}(\varphi) = \mathbf{f}$.

We now investigate the case where φ is the formula $\textsc{sum}\{(n, \overline{x}) \mid \psi\} \leq y$. Let $\tilde{I}$ be a three-valued structure and θ a variable assignment. According to Definition 3.15, $\tilde{I}\theta(\varphi) = \mathbf{t}$ iff $\tilde{I}\theta(\textsc{sum}\{(n, \overline{x}) \mid \psi\})_{\max}$ is less than $\theta(y)$, i.e., if the maximal value of the sum of elements in a set approximated by $I\theta(\{(n, \overline{x}) \mid \psi\})$ is less than $\theta(y)$. It is straightforward to see that if $\tilde{V}$ is a three-valued set, the maximal value in $\{\textsc{sum}(v) \mid v \sqsubset \tilde{V}\}$ is given by

$$\left(\sum_{(n,\overline{d})^{\mathbf{t}} \in \tilde{V}} n\right) + \left(\sum_{(n,\overline{d})^{\mathbf{u}} \in \tilde{V} \text{ and } n>0} n\right).$$

We also have the following equalities:

$$\begin{aligned}
&\left(\sum_{(n,\overline{d})^{\mathbf{t}}\in\tilde{V}} n\right) + \left(\sum_{(n,\overline{d})^{\mathbf{u}}\in\tilde{V} \text{ and } n>0} n\right)\\
&= \left(\sum_{(n,\overline{d})^{\mathbf{t}}\in\tilde{V} \text{ and } n<0} n\right) + \left(\sum_{(n,\overline{d})^{\mathbf{t}}\in\tilde{V} \text{ and } n>0} n\right) + \left(\sum_{(n,\overline{d})^{\mathbf{u}}\in\tilde{V} \text{ and } n>0} n\right)\\
&= \left(\sum_{(n,\overline{d})^{\mathbf{t}}\in\tilde{V} \text{ and } n<0} n\right) + \left(\sum_{(n,\overline{d})^{\mathbf{t}}\in\tilde{V} \text{ and } n>0} n\right)\\
&+ \left(\sum_{(n,\overline{d})^{\mathbf{v}}\in\tilde{V},\mathbf{v}\in\{\mathbf{t},\mathbf{f},\mathbf{u}\} \text{ and } n>0} n\right) - \left(\sum_{(n,\overline{d})^{\mathbf{t}}\in\tilde{V} \text{ and } n>0} n\right) - \left(\sum_{(n,\overline{d})^{\mathbf{f}}\in\tilde{V} \text{ and } n>0} n\right)\\
&= \left(\sum_{(n,\overline{d})^{\mathbf{t}}\in\tilde{V} \text{ and } n<0} n\right) + \left(\sum_{(n,\overline{d})^{\mathbf{v}}\in\tilde{V},\mathbf{v}\in\{\mathbf{t},\mathbf{f},\mathbf{u}\} \text{ and } n>0} n\right) - \left(\sum_{(n,\overline{d})^{\mathbf{f}}\in\tilde{V} \text{ and } n>0} n\right)
\end{aligned}$$

It follows that for a two-valued structure I, $I\theta(\varphi) = \mathbf{t}$ iff $I\theta$ satisfies the formula

$$\textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{ct}} \wedge n < 0\} + \textsc{sum}\{(n,\overline{x}) \mid n > 0\} - \textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{cf}} \wedge n > 0\} \leq y.$$

Converting this formula to TNF produces the formula

$$\begin{aligned}
&\exists x_1 \exists x_2 \exists x_3\ (\\
&\quad (\textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{ct}} \wedge n < 0\} \leq x_1) \wedge (\textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{ct}} \wedge n < 0\} \geq x_1)\\
&\quad \wedge (\textsc{sum}\{(n,\overline{x}) \mid n > 0\} \leq x_2) \wedge (\textsc{sum}\{(n,\overline{x}) \mid n > 0\} \geq x_2)\\
&\quad \wedge (\textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{cf}} \wedge n > 0\} \leq x_3) \wedge (\textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{cf}} \wedge n > 0\} \geq x_3)\\
&\quad \wedge x_1 + x_2 - x_3 \leq y),
\end{aligned}$$

It is straightforward to check that this sentence is logically equivalent to the monotone sentence

$$\begin{aligned}
&\exists x_1 \exists x_2 \exists x_3\ (\\
&\quad \textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{ct}} \wedge n < 0\} \leq x_1\\
&\quad \wedge \textsc{sum}\{(n,\overline{x}) \mid n > 0\} \leq x_2\\
&\quad \wedge \textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{cf}} \wedge n > 0\} \geq x_3\\
&\quad \wedge x_1 + x_2 - x_3 \leq y).
\end{aligned} \tag{A.19}$$

Hence we define

$$(\textsc{sum}\{(n,\overline{x}) \mid \psi\} \leq y)^{\mathrm{ct}} := \text{(A.19)}.$$

Observe that under the assumption that all domain elements of subsorts of $\mathfrak{Int}$ are strictly larger than zero, formula (A.19) is equivalent to the formula

$$\exists x_2 \exists x_3\ (\textsc{sum}\{(n,\overline{x}) \mid \top\} \leq x_2 \wedge \textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{cf}}\} \geq x_3 \wedge x_2 - x_3 \leq y),$$

which is exactly the formula given in Table 4.3.
A similar reasoning leads to the formula

$$\begin{aligned}\exists x_1 \exists x_2 \exists x_3 \ (& \\ & \textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{ct}} \wedge n > 0\} \geq x_1 \\ & \wedge \textsc{sum}\{(n,\overline{x}) \mid n < 0\} \geq x_2 \\ & \wedge \textsc{sum}\{(n,\overline{x}) \mid \psi^{\mathrm{cf}} \wedge n < 0\} \leq x_3 \\ & \wedge x_1 + x_2 - x_3 \geq y). \end{aligned} \tag{A.20}$$

as definition for $(\textsc{sum}\{(n,\overline{x}) \mid \psi\} \geq y)^{\mathrm{ct}}$.

Product aggregate

It is possible — but beyond the scope of this thesis — to construct for a formula φ of the form $\textsc{prod}\{(n,\overline{x}) \mid \psi\} \leq y$ or $\textsc{prod}\{(n,\overline{x}) \mid \psi\} \geq y$ two formulas φ^{ct} and φ^{cf} such that for every three-valued structure $\tilde{I}$, $\tilde{I}(\varphi) = \tilde{I}^{\mathrm{tf}}(\varphi^{\mathrm{ct}}, \varphi^{\mathrm{cf}})$ and for every three-valued structure $\tilde{J} \geq_p \tilde{I}$, $\tilde{J}^{\mathrm{tf}}(\varphi^{\mathrm{ct}}) \geq_t \tilde{I}^{\mathrm{tf}}(\varphi^{\mathrm{ct}})$ and $\tilde{J}^{\mathrm{tf}}(\varphi^{\mathrm{cf}}) \geq_t \tilde{I}^{\mathrm{tf}}(\varphi^{\mathrm{cf}})$. However, since for a three-valued set $\tilde{V}$, calculating the minimal and maximal value in $\{\textsc{prod}(v) \mid v \sqsubset \tilde{V}\}$ is more complicated than calculating the minimal and maximal value in $\{\textsc{sum}(v) \mid v \sqsubset \tilde{V}\}$, the desired formulas φ^{ct} and φ^{cf} are much longer and more complicated than, e.g., the formula (A.19). We therefore doubt the practical importance of these formulas. Moreover, the practical examples in the field of knowledge representation that actually involve a product aggregate are scarce and we are not aware of any example in the literature that involves a product aggregate over a three-valued set containing negative numbers.

Cardinality, minimum and maximum

For the formulas φ of the form $\mathrm{F}\{(n,\overline{x}) \mid \psi\}$ where $\mathrm{F} \in \{\textsc{card}, \textsc{min}, \textsc{max}\}$, the formulas φ^{ct} given in Table 4.3 remain correct for structures that involve negative numbers, i.e., for these formulas Proposition 4.22 generalizes to arbitrary finite structures.

Proof of Theorem 6.2

In the following, we say that a structure M satisfies a branch B of an MX-tree if $M \models \varphi[\overline{x}/\overline{d}]$ for each signed instance $\varphi[\overline{x}/\overline{d}]^{\oplus} \in B$ and $M \models \neg\varphi[\overline{x}/\overline{d}]$ for each signed instance $\varphi[\overline{x}/\overline{d}]^{\ominus} \in B$. We denote this by $M \models B$.

Lemma A.16. *If $M \models_{\tilde{I}} T$ and $\mathcal{T}$ is an MX-tree for $\langle T, \tilde{I}\rangle$ then there is a branch B of $\mathcal{T}$ such that $M \models B$.*

Proof. We prove the lemma by induction. Clearly, if $\mathcal{T}$ is the empty MX-tree, then M satisfies a branch, namely the empty branch, of $\mathcal{T}$. Now assume M

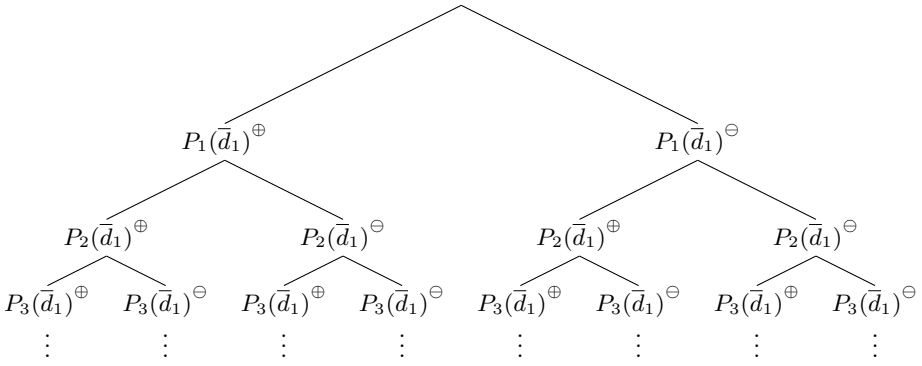


Figure A.1: Begin of the MX-tree $\mathcal{T}'$

satisfies a branch B' of an MX-tree $\mathcal{T}'$ and $\mathcal{T}$ is obtained from $\mathcal{T}'$ by applying an MX-rule. It follows from the soundness of MX-rules that there is at least one branch B of $\mathcal{T}$ such that $B' \subseteq B$ and $M \models B$. ■

Lemma A.17. *If there exists an MX-proof for $\langle T, \tilde{I}\rangle$, then there is no $M \geq_p \tilde{I}$ that satisfies T.*

Proof. Let $\mathcal{T}$ be an MX-proof for $\langle T, \tilde{I}\rangle$ and assume towards a contradiction that $M \models_{\tilde{I}} T$. Because of the previous lemma, there exists a branch B of $\mathcal{T}$ such that $M \models B$. Because all branches in $\mathcal{T}$ are closed, there exists an instance $\varphi[\overline{x}/\overline{d}]$ such that both $\varphi[\overline{x}/\overline{d}]^{\oplus} \in B$ and $\varphi[\overline{x}/\overline{d}]^{\ominus} \in B$. Therefore $M \models \varphi[\overline{x}/\overline{d}]$ and $M \models \neg\varphi[\overline{x}/\overline{d}]$. Contradiction. ■

Lemma A.18. *If there exists no $M \geq_p \tilde{I}$ that satisfies T, then there exists an MX-proof for $\langle T, \tilde{I}\rangle$.*

Proof. Denote the vocabulary of T by Σ. We prove this lemma in the case Σ does not contain function symbols. The more general case can be proven in a similar way.
An MX-tree $\mathcal{T}$ for $\langle T, \tilde{I}\rangle$ can be constructed as follows. Let $P_1(\overline{d}_1)$, $P_2(\overline{d}_2)$, $\ldots$, $P_n(\overline{d}_n)$ be all domain atoms over Σ and the domain D of $\tilde{I}$. Start the construction of $\mathcal{T}$ by applying the cut-rule on $P_1(\overline{d}_1)$. Next, apply the cut-rule for $P_2(\overline{d}_2)$ on each of the branches. Then apply the cut-rule for $P_3(\overline{d}_3)$, etc. The resulting tree is shown in Figure A.1. Denote this tree by $\mathcal{T}'$. For each Σ-structure M with domain D, there is exactly one branch B of $\mathcal{T}'$ such that $M \models B$.
If $M \not\geq_p \tilde{I}$ then there is a domain atom $P(\overline{d})$ such that either $M \models P(\overline{d})$ while $\tilde{I}(P(\overline{d})) = \mathbf{f}$, or $M \models \neg P(\overline{d})$ while $\tilde{I}(P(\overline{d})) = \mathbf{t}$. In the former case, the branch B of $\mathcal{T}'$ satisfied by M contains $P(\overline{d})^{\oplus}$, in the latter B contains $P(\overline{d})^{\ominus}$. It follows that in both cases the branch B can be closed by applying an initialization rule.
Now assume $M \geq_p \tilde{I}$ and let B be the branch of $\mathcal{T}'$ satisfied by M. We show that also in this case, B can be closed by applying MX-rules different from the

cut rule. We first prove by induction that for every instance φ of a formula over Σ that is false in M, $\varphi^{\ominus}$ can be added to B. The base case is trivial: if φ is an atom P, it follows that $P^{\ominus}$ already belongs to B. We prove one of the inductive cases, the others are similar. Let φ be the instance $\psi_1(\overline{d}_1) \wedge \psi_2(\overline{d}_2)$. Then either $M \not\models \psi_1(\overline{d}_1)$ or $M \not\models \psi_2(\overline{d}_2)$. Assume the former. Then from the induction hypothesis, it follows that $\psi_1(\overline{d}_1)^{\ominus}$ can be added to B. Next, rule ($\wedge$-$\uparrow$) can be used to add $(\psi_1(\overline{d}_1) \wedge \psi_2(\overline{d}_2))^{\ominus}$.
Since $M \not\models_{\tilde{I}} T$, there is a sentence φ of T that is not satisfied in M. Hence $\varphi^{\ominus}$ can be added to B. By initialization rule (I+$\downarrow$), also $\varphi^{\oplus}$ can be added, which closes the branch.
We conclude that $\mathcal{T}'$ can be extended to an MX-proof for $\langle T, \tilde{I}\rangle$. ■

Proof. (of Theorem 6.2) Follows directly from Lemma A.17 and Lemma A.18. ■

Proof of Lemma 6.3

First observe that $\tilde{B}$ is always two-valued. Indeed, if it would assign $\mathbf{u}$ to an atom $P(\overline{d})$, then neither $P(\overline{d})^{\oplus}$ nor $P(\overline{d})^{\ominus}$ occurs in B, and it follows that B cannot be saturated since the T-restricted cut-rule

$$\frac{}{P(\overline{d})^{\oplus} \mid P(\overline{d})^{\ominus}}$$

could be applied to extend it.
In the proof of Lemma A.18, we have shown that if $\tilde{I} \not\leq_p \tilde{B}$ or $\tilde{B} \not\models T$, B can be closed by applying only T-restricted MX-rules. Since B is saturated, it follows that $\tilde{B} \models T$.

Proof of Proposition 6.5

We only give a sketch of the proof of Proposition 6.5, since the full proof involves many details but is rather straightforward and does not provide additional insight compared to the proof sketch below. We split the proof sketch over several lemmas. In all lemmas, T' denotes the ENF theory obtained by rewriting T.

Lemma A.19. *If Aux is an auxiliary predicate introduced when rewriting T to T' and $\forall \overline{x}\ (Aux(\overline{x}) \Leftrightarrow \psi)$ is a sentence of T', then $\varphi_{Aux} = \lceil \psi \rceil$.*

Proof. The lemma is a direct result of Algorithm 4.1. ■

Lemma A.20. *Let B be a branch of an MX-tree $\mathcal{T}$ for $\langle T, \tilde{I}\rangle$ and $\tilde{J}$ a structure over the vocabulary of T' such that*

- *$\lceil P(\overline{d}) \rceil^{\oplus}$ and $\lceil \neg P(\overline{d}) \rceil^{\ominus}$ belong to B for every domain atom $P(\overline{d})$ such that $P^{\tilde{J}}(\overline{d}) \geq_p \mathbf{t}$;*

- $\lceil P(\overline{d})\rceil^{\ominus}$ *and* $\lceil \neg P(\overline{d})\rceil^{\oplus}$ *belong to* B *for every domain atom* $P(\overline{d})$ *such that* $P^{\tilde{J}}(\overline{d}) \geq_p \mathbf{f}$.

If $\forall \overline{x}\ (\varphi \Rightarrow L[\overline{x}]) \in \mathrm{INF}(T')$ *and* $\tilde{J}[\overline{x}/\overline{d}](\varphi) \geq_p \mathbf{t}$*, then the result of adding* $L[\overline{x}/\overline{d}]^{\oplus}$ *to* B *in* $\mathcal{T}$ *is an MX-tree for* $\langle T, \tilde{I}\rangle$.

Proof. The proof of the lemma consists of a analysis of all cases in Table 4.1. We prove two of the cases. All others are similar.
For the first case, assume φ is the formula $P(\overline{x})$ and $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow L[\overline{x}])$ is a sentence of T'. Since $\tilde{J}[\overline{x}/\overline{d}](P(\overline{x})) \geq_p \mathbf{t}$, it follows that $P(\overline{d})^{\oplus} \in B$. If P is an auxiliary predicate, then $\varphi_P = \lceil L[\overline{x}]\rceil$. Hence $\lceil L[\overline{x}/\overline{d}]\rceil^{\oplus}$ already belongs to B. Adding it again to B in $\mathcal{T}$ obviously results in an MX-tree.
For the second case, assume φ is the formula $(P(\overline{z}) \wedge (\forall y'\ (y \neq y' \Rightarrow \neg L[\overline{z}, y'])))$, $\overline{x} = (\overline{z}, y)$, $\overline{d} = (\overline{d}_z, d_y)$ and $\forall \overline{z}\ (P(\overline{z}) \Leftrightarrow \exists y\ L[\overline{x}])$ is a sentence of T'. Since $\tilde{J}[\overline{x}/\overline{d}](\varphi) \geq \mathbf{t}$, it follows that $\tilde{J}[\overline{z}/\overline{d}_z](P(\overline{z})) \geq_p \mathbf{t}$ and $\tilde{J}[\overline{z}/\overline{d}_z][y'/d_y](L[\overline{z}, y']) \geq_p \mathbf{f}$ for every $d'_y \neq d_y$. It follows that $\lceil P(\overline{d}_z)\rceil^{\oplus} \in B$ and $L[\overline{z}/\overline{d}_z][y'/d'_y]^{\ominus} \in B$ for every $d'_y \neq d_y$. Because $\lceil P(\overline{d}_z)\rceil^{\oplus} = \lceil \exists y\ L[\overline{x}]\rceil^{\oplus}$, it follows that propagation rule $(\exists{+}{\downarrow})$ can be applied to add $L[\overline{d}]^{\oplus}$ to B and obtain an MX-tree. ■

The adaption of Algorithm 4.4 to generate MX-trees is shown in Algorithm A.1. Proposition 6.5 can now be proven by showing that the conditions on $\mathcal{T}$, B and $\tilde{J}$ as in Lemma A.20 hold at each moment during the execution of the algorithm. Because all initialization rules are applied before propagation starts (line 1), the conditions hold before propagation is executed the first time. Lines 2–5 of function `expandTree` ensure that the conditions are preserved when applying propagation. Lines 13–14 do the same when the cut rule is applied. Lemma A.20 guarantees that an MX-tree is build. It is straightforward to see that if all branches of this tree are closed, a strictly four-valued structure would have been returned by the original model generation algorithm.

Algorithm A.1: Generating an MX-tree

Input: A theory T and structure $\tilde{I}$
1 $\mathcal{T}$:= the tree obtained by applying all initialization rules on $\langle T, \tilde{I}\rangle$;
2 B := the only branch of $\mathcal{T}$;
3 T' := INF(T);
4 $\tilde{J}$:= the least precise extension of $\tilde{I}$ to the vocabulary of T';
5 $(\tilde{J}, \mathcal{T})$:= `expandTree(`T'`,`$\tilde{J}$`,`$\mathcal{T}$`,`B`)`;
6 **return** $\mathcal{T}$;

Function `expandTree`$(T', \tilde{J}, \mathcal{T}, B)$

while *there exists a* $\varphi \in T'$ *such that* $\mathscr{I}^{\varphi}(\tilde{J}) \neq \tilde{J}$ **do**
  for *every domain atom* $P(\overline{d})$ *such that* $P^{\tilde{J}}(\overline{d}) <_t P^{\mathscr{I}^{\varphi}(\tilde{J})}(\overline{d})$ **do**
    Extend $\mathcal{T}$ by adding $\lceil P(\overline{d}) \rceil^{\oplus}$ and $\lceil \neg P(\overline{d}) \rceil^{\ominus}$ to B;
  for *every domain atom* $P(\overline{d})$ *such that* $P^{\tilde{J}}(\overline{d}) >_t P^{\mathscr{I}^{\varphi}(\tilde{J})}(\overline{d})$ **do**
    Extend $\mathcal{T}$ by adding $\lceil P(\overline{d}) \rceil^{\ominus}$ and $\lceil \neg P(\overline{d}) \rceil^{\oplus}$ to B;
  $\tilde{J} := \mathscr{I}^{\varphi}(\tilde{J})$;
if $\tilde{J}$ *is two-valued* **then**
  return $(\tilde{J}, \mathcal{T})$;
else if $\tilde{J}$ *is four-valued* **then**
  return $(\top^{\leq_p}, \mathcal{T})$;
else
  $P(\overline{d})$:= `choose`$(\tilde{J})$;
  Extend $\mathcal{T}$ by applying the cut rule on B to add $\lceil P(\overline{d}) \rceil^{\oplus}$ and $\lceil P(\overline{d}) \rceil^{\ominus}$;
  Apply the negation rules to add $\lceil \neg P(\overline{d}) \rceil^{\ominus}$ below $\lceil P(\overline{d}) \rceil^{\oplus}$ and $\lceil \neg P(\overline{d}) \rceil^{\oplus}$ below $\lceil P(\overline{d}) \rceil^{\ominus}$;
  B' := the branch of $\mathcal{T}$ that ends in $\lceil \neg P(\overline{d}) \rceil^{\ominus}$;
  B'' := the branch of $\mathcal{T}$ that ends in $\lceil \neg P(\overline{d}) \rceil^{\oplus}$;
  $(\tilde{K}, \mathcal{T})$:= `expandTree`$(T', \tilde{J}[P(\overline{d})/\mathbf{t}], \mathcal{T}, B')$;
  if $\tilde{K}$ *is two-valued* **then**
    return $(\tilde{K}, \mathcal{T})$;
  else
    return `expandTree`$(T', \tilde{J}[P(\overline{d})/\mathbf{f}], \mathcal{T}, B'')$;

Appendix B

Unit propagation

Several times in this thesis we pointed out that there is a relation between *unit propagation* and the propagation method presented in Chapter 4. In this section, we examine this relation in detail and present some of its practical consequences. All theories in this section are FO theories.

Unit propagation versus INF propagation

Unit propagation is the main form of propagation applied in current SAT solvers. Formally, it is defined as follows.

Definition B.1. Let C be the clause $L_1 \vee \ldots \vee L_n$ over propositional vocabulary Σ. The *unit propagator* $\mathcal{U}^C$ *associated to* C is the operator on Σ-structures defined by

$$L^{\mathcal{U}^C(\tilde{I})} = \begin{cases} \mathrm{lub}_{\leq_p}(\mathbf{t}, L^{\tilde{I}}) & \text{if } L \text{ belongs to } C \text{ and} \\ & (L')^{\tilde{I}} \geq_p \mathbf{f} \text{ for every other literal } L' \text{ in } C; \\ L^{\tilde{I}} & \text{otherwise.} \end{cases}$$

for every literal L and structure $\tilde{I}$.

Proposition B.1. *$\mathcal{U}^C$ is a monotone propagator for every clause C.*

If T is a propositional theory in CNF, we denote by $\mathcal{U}(T)$ the set $\{\mathcal{U}^C \mid C \in T\}$, i.e., the set of the unit propagators associated to the clauses in T. Since all these propagators are monotone, Proposition 4.3 ensures the propagator $\lim_{\mathcal{U}(T)}$ is well-defined. The DPLL algorithm (Davis and Putnam, 1960; Davis et al., 1962) — the algorithm underlying many of today's most efficient SAT solvers — is basically Algorithm 4.4, where the function `propagate` is implemented by $\lim_{\mathcal{U}(T)}$.
Let T be a theory over vocabulary Σ and D a finite domain. Denote by T_1 the ENF theory obtained by applying Algorithm 4.1 to T and let T_2 be the propositional theory obtained by applying the Tseitin transformation to $\mathrm{Gr}_{\mathrm{full}}(T, D)$.

We show that the propagation method we presented in Chapter 4 coincides with unit propagation in the sense that $(\lim_{\mathscr{I}(\mathrm{INF}(T_1))}(\tilde{I}))|_\Sigma = (\lim_{\mathcal{U}(T_2)}(\tilde{I}))|_\Sigma$ for every structure $\tilde{I}$ with domain D. We prove this property in two steps. First we show that unit propagation on the grounding T_3 of T_1 yields the same results as $\lim_{\mathscr{I}(\mathrm{INF}(T_1))}$. Next, we show that unit propagation on T_3 coincides with unit propagation on T_1.
For a propositional formula φ of the form $L \Leftrightarrow L_1 \vee \ldots \vee L_n$, where L, L_1, …, L_n are literals, we denote by $\mathrm{CNF}(\varphi)$ the equivalent propositional CNF theory

$$\{\neg L_1 \vee L, \ldots, \neg L_n \vee L, \neg L \vee L_1 \vee \ldots \vee L_n\}.$$

Similarly, if φ is the formula $L \Leftrightarrow L_1 \wedge \ldots \wedge L_n$, then $\mathrm{CNF}(\varphi)$ denotes the CNF theory

$$\{L_1 \vee \neg L, \ldots, L_n \vee \neg L, L \vee \neg L_1 \vee \ldots \vee \neg L_n\}.$$

This notation is pointwise extended to theories consisting of only (conjunctions of) sentences of the form $L \Leftrightarrow L_1 \vee \ldots \vee L_n$ or $L \Leftrightarrow L_1 \wedge \ldots \wedge L_n$. It is straightforward to check that for each ENF sentence φ, the full grounding of φ is such a conjunction of sentences of the form $L \Leftrightarrow L_1 \vee \ldots \vee L_n$ or $L \Leftrightarrow L_1 \wedge \ldots \wedge L_n$. The next proposition states that applying unit propagation on the grounding of an ENF sentence yields the same result as applying the INF propagators associated to φ.

Proposition B.2. *For every ENF sentence φ, finite domain D and structure $\tilde{I}$ with domain D, $\lim_{\mathscr{I}(\mathrm{INF}(\varphi))}(\tilde{I}) = \lim_{\mathcal{U}(\mathrm{CNF}(Gr_{full}(\varphi,D)))}(\tilde{I})$.*

Proof. We prove the case where φ is of the form $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \forall y\ Q(\overline{x}, y))$. All other cases can be proven in a similar fashion.
If φ is the sentence $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \forall y\ Q(\overline{x}, y))$, then $\mathrm{INF}(\varphi)$ is the set of sentences

$$\forall \overline{x} \forall y\ (P(\overline{x}) \Rightarrow Q(\overline{x}, y)), \tag{B.1}$$
$$\forall \overline{x}\ ((\neg \forall y\ Q(\overline{x}, y)) \Rightarrow \neg P(\overline{x})), \tag{B.2}$$
$$\forall \overline{x}\ ((\forall y\ Q(\overline{x}, y)) \Rightarrow P(\overline{x})), \tag{B.3}$$
$$\forall \overline{x} \forall y\ (\neg P(\overline{x}) \wedge (\forall y'\ (y \neq y' \Rightarrow Q(\overline{x}, y'))) \Rightarrow \neg Q(\overline{x}, y)). \tag{B.4}$$

The propositional CNF theory $\mathrm{CNF}(\mathrm{Gr}_{\mathrm{full}}(\varphi, D))$ is given by

$$\left(\bigcup_{\overline{d} \in D^{|\overline{x}|}} P(\overline{d}) \vee \left(\bigvee_{d' \in D} \neg Q(\overline{d}, d')\right)\right) \cup \left(\bigcup_{\overline{d} \in D^{|\overline{x}|}, d' \in D} \neg P(\overline{d}) \vee Q(\overline{d}, d')\right). \tag{B.5}$$

Let $\tilde{I}$ be a structure with domain D. To complete the proof, it suffices to show that

- for every domain literal L such that $\tilde{I}(L) = \mathbf{u}$, there exists a $\psi \in \mathrm{INF}(\varphi)$ such that $\mathscr{I}^\psi(\tilde{I})(L) = \mathbf{t}$ iff there exists a clause $C \in \mathrm{CNF}(\mathrm{Gr}_{\mathrm{full}}(\varphi, D))$ such that $\mathcal{U}^C(\tilde{I})(L) = \mathbf{t}$ and

- for every domain literal L such that $\tilde{I}(L) = \mathbf{f}$, there exists a $\psi \in \mathrm{INF}(\varphi)$ such that $\mathscr{I}^{\psi}(\tilde{I})(L) = \mathbf{i}$ iff there exists a clause $C \in \mathrm{CNF}(\mathrm{Gr}_{\mathrm{full}}(\varphi, D))$ such that $\mathcal{U}^{C}(\tilde{I})(L) = \mathbf{i}$.

We prove the first of these statements. The proof of the second one is completely similar.
First assume L is the literal $P(\overline{d})$ and $P^{\tilde{I}}(\overline{d}) = \mathbf{u}$. The only INF propagator associated to φ that can possibly make $P(\overline{d})$ true is the propagator $\mathscr{I}^{(\mathrm{B}.3)}$, the only clause in φ's grounding whose unit propagation can make $P(\overline{d})$ true is the clause $C = P(\overline{d}) \vee \big(\bigvee_{d' \in D} \neg Q(\overline{d}, d')\big)$. It follows that $\mathscr{I}^{(\mathrm{B}.3)}(\tilde{I})(P(\overline{d})) = \mathbf{t}$ iff $\tilde{I}(Q(\overline{d}, d')) \geq_p \mathbf{t}$ for all $d' \in D$ iff $\mathcal{U}^{C}(\tilde{I})(P(\overline{d})) = \mathbf{t}$.
Now assume L is the literal $\neg P(\overline{d})$ and $P^{\tilde{I}}(\overline{d}) = \mathbf{u}$. The only INF propagator that can make $P(\overline{d})$ false is $\mathscr{I}^{(\mathrm{B}.2)}$, the only clauses whose unit propagation can make $P(\overline{d})$ false are the clauses of the form $\neg P(\overline{d}) \vee Q(\overline{d}, d')$. $P(\overline{d})$ is false in $\mathscr{I}^{(\mathrm{B}.2)}(\tilde{I})$ iff there exists a d' such that $\tilde{I}(Q(\overline{d}, d')) \geq_p \mathbf{f}$ iff there exists a clause $C = \neg P(\overline{d}) \vee Q(\overline{d}, d')$ such that $P(\overline{d})$ is false in $\mathcal{U}^{C}(\tilde{I})$.
The cases where L is the literal $Q(\overline{d}, d')$ or $\neg Q(\overline{d}, d')$ can be proven in a similar fashion. ■

Corollary B.3. *For every ENF theory T, finite domain D and structure $\tilde{I}$ with domain D, $\lim_{\mathscr{I}(\mathrm{INF}(T))}(\tilde{I}) = \lim_{\mathcal{U}(\mathrm{CNF}(Gr_{full}(T,D)))}(\tilde{I})$.*

Now we come to the second part of the proof the propagation method of Chapter 4 coincides with unit propagation. The next proposition expresses that for a theory T and corresponding ENF theory T', unit propagation on the grounding of T corresponds to unit propagation on the grounding of T'.

Proposition B.4. *Let T be a theory over vocabulary Σ and let T' be the ENF theory obtained from T by applying Algorithm 4.1. Let D be a finite domain and let T_g be the propositional CNF theory obtained by applying the Tseitin transformation on $Gr_{full}(T, D)$. Then $\lim_{\mathcal{U}(\mathrm{CNF}(Gr_{full}(T',D)))}(\tilde{I})|_{\Sigma} = \lim_{\mathcal{U}(T_g)(\tilde{I})}|_{\Sigma}$ for every structure $\tilde{I}$ with domain D.*

Proof idea. The proof consists of a careful analysis of Algorithm 4.1: for each step the algorithm takes to transform a theory T_1 in theory T_2, it is checked that unit propagation on T_1's grounding corresponds to unit propagation on T_2's grounding. It is outside the scope of this thesis to provide the (very long) proof. ■

Combining Corollary B.3 and Proposition B.4 yields the correspondence between the propagation method of Chapter 4 on a theory T and unit propagation on the grounding of T.

Theorem B.5. *Let Σ, T, T', D and T_g be as in Proposition B.4. Then for every structure $\tilde{I}$ with domain D, $(\lim_{\mathscr{I}(\mathrm{INF}(T'))}(\tilde{I}))|_{\Sigma} = (\lim_{\mathcal{U}(T_g)}(\tilde{I}))|_{\Sigma}$.*

Implications

Unit propagation (and any other propositional propagation method) can be used as a method to simplify a CNF theory. Basically, if a literal L is derived to be true by unit propagation on T, T can be simplified by deleting every clause containing L, as well as any occurrence of $\neg L$ in clauses of T. Theorem B.5 implies that this simplification method is as strong as using bounds to simplify a grounding.

All DPLL-based SAT solvers apply unit propagation before a first choice is made, i.e., before line 5 is reached in Algorithm 4.4. As such, they apply unit propagation to (at least implicitly) simplify their input theory in the way described above. Since unit propagation can be implemented very efficiently, it is not beneficial for a model generator that relies on grounding to spend much time in computing c-maps that only yield slightly smaller groundings but not significantly faster grounding. Indeed, the same reduction in size will be obtained much more efficiently by the unit propagation of the applied propositional solver.

Another implication of Theorem B.5 is that models for ground theories T_g produced by grounding with bounds are computed as efficiently as those for ground theories T'_g produced by grounding without bounds. Indeed, after the initial unit propagation of a SAT solver, T_g and T'_g are simplified to similar theories. The only difference is that the simplified theory obtained from T'_g may contain some unnecessary auxiliary symbols that were introduced by applying the Tseitin transformation and that do not occur in the simplified theory obtained from T_g. Since grounding with bounds is almost always faster than grounding without bounds, we conclude that *model generation* with bounds is almost always faster than model generation without bounds.

Corollary B.3 implies that for an ENF theory T, propagation using the propagators in $\mathscr{I}(\text{INF}(T))$ can be mimicked by grounding T and applying unit propagation. The proof of proposition B.2 indicates that for each possible result of applying unit propagation on a clause C in the grounding of T, there exists an INF sentence φ in $\text{INF}(T)$ such that applying $\mathscr{I}^{\varphi}$ produces the same result. It follows that the trace of running a SAT solver that implements the DPLL algorithm on the grounding of T can be translated in a trace of running Algorithm 4.4 on T. From Proposition 6.5, it now follows that MX-proofs for an input theory T can be constructed from the trace of a DPLL SAT solver ran on the grounding of T.

Current efficient SAT solvers extend the DPLL algorithm with sophisticated heuristics, backjumping instead of chronological backtracking, clause learning and restarts. A good overview is presented by Mitchell (2005). SAT solvers implementing these techniques on top of DPLL are often called *conflict driven clause learning* (CDCL) SAT solvers. It can be shown that a trace of a CDCL SAT solver can be translated to a trace of a DPLL SAT solver, which indicates that CDCL SAT solvers can be used to efficiently produce MX-proofs.

Appendix C

Implementation details of GIDL

In this appendix, we provide details on the implementation of GIDL.

Input syntax

The input language of GIDL is full FO(·), as described in Chapter 3. The syntax of GIDL's input language is an ASCII version of FO(·). Table C.1 describes for some of the symbols of FO(·) the corresponding ASCII symbols in GIDL. Listing C.1 presents the complete battleship puzzle in GIDL's input language. As one can see by comparing Example 2.3, Example 2.6 and Example 3.20 with Listing C.1, GIDL's input is close to FO(·) as described in this thesis. Also observe that it is not necessary to specify the sorts of each variable in GIDL's input. As far as no ambiguity is possible, the sorts of variables are derived automatically. The full syntax of GIDL's input language is described in the user's manual of IDP (Wittocx and Mariën, 2008).

FO(·)	$\wedge$	$\vee$	$\neg$	$\forall$	$\exists$	$\Rightarrow$	$\Leftrightarrow$	$\leftarrow$	$=$	$\neq$
ASCII	`&`	`|`	`~`	`!`	`?`	`=>`	`<=>`	`<-`	`=`	`~=`

Table C.1: From FO(·) to ASCII

Listing C.1: battleship puzzle

```
/** Battleship puzzle specification **/

%% vocabulary

   // Sorts
```

```
   type int Row
   type int Col
   type int Num
   type int Length
   type Direction
   type Ship

   // Predicates
   ContainsShip(Row,Col)
   ContainsShip(Row,Col,Ship)
   Neighbour(Row,Col,Row,Col)

   // Functions
   RowNum(Row) : Num
   ColNum(Col) : Num
   Length(Ship) : Length
   Horizontal : Direction
   Vertical   : Direction
   InitialRow(Ship) : Row
   InitialCol(Ship) : Col
   ShipDir(Ship)  : Direction

%% theory

   { Neighbour(r1,c1,r2,c2) <- abs(r2-r1) =< 1 &
                               abs(c2-c1) =< 1.}

   { ContainsShip(r,c,s) <- InitialRow(s) = r &
                            InitialCol(s) = ic &
                            ShipDir(s) = Horizontal &
                            c =< ic < c + Length(s).
     ContainsShip(r,c,s) <- InitialRow(s) = ir &
                            InitialCol(s) = c &
                            ShipDir(s) = Vertical &
                            r =< ir < r + Length(s).
   }

   { ContainsShip(r,c) <- ContainsShip(r,c,s).}

   ! r : card{ c : ContainsShip(r,c) } = RowNum(r).
   ! c : card{ r : ContainsShip(r,c) } = ColNum(c).

   ! r1 c1 r2 c2 s1 s2 : Neighbour(r1,c1,r2,c2) &
                         ContainsShip(r1,c1,s1) &
                         ContainsShip(r2,c2,s2)
                         => s1 = s2.
```

```
/** Input and output vocabulary **/

%% inputvoc
   RowNum/1:
   ColNum/1:
   Length/1:
   Horizontal/0:
   Vertical/0:

%% outputvoc
   InitialCol/1:
   InitialRow/1:
   ShipDir/1:


/** Input structure **/
%% structure

   Row = {1..10}
   Col = {1..10}
   Num = {0..10}
   Length = {1..4}
   Direction = { Horizontal; Vertical }
   Horizontal = Horizontal
   Vertical = Vertical
   Ship = { Submarine1; Submarine2; Submarine3;
            Submarine4; Destroyer1; Destroyer2;
            Destroyer3; Cruiser1; Cruiser2; Battleship}
   Length = { Submarine1 -> 1; Submarine2 -> 1;
              Submarine3 -> 1; Submarine4 -> 1;
              Destroyer1 -> 2; Destroyer2 -> 2;
              Destroyer3 -> 2; Cruiser1 -> 3;
              Cruiser2 -> 3; Battleship -> 4 }
   RowNum = { 1 -> 2; 2 -> 1; 3 -> 5; 4 -> 0; 5 -> 2;
              6 -> 2; 7 -> 1; 8 -> 4; 9 -> 1; 10 -> 2 }
   ColNum = { 1 -> 0; 2 -> 1; 3 -> 2; 4 -> 3; 5 -> 3;
              6 -> 1; 7 -> 3; 8 -> 0; 9 -> 6; 10 -> 1 }

%% partial
   ContainsShip = {2,3; 3,3}{1,3; 9,5}
```

Preprocessing

Before the actual grounding starts, GIDL rewrites an input theory and structure into a normal form resembling TNF. Specifically, an input theory T over vocabulary Σ and input structure $\tilde{I}$ is rewritten to a theory T' and two-valued structure J over a vocabulary σ such that

1. all negations in T' occur directly in front of atoms;
2. every function symbol F such that $\tilde{I}$ is strictly three-valued on F only occurs in T' as outermost symbol in atoms of the form $F(\bar{t}) = t'$;
3. all aggregate expressions in T' are of the form $\mathrm{F}(V) \geq t$ or $\mathrm{F}(V) \leq t$;
4. if T' contains a definition, this definition depends on predicates with a strictly three-valued interpretation in $\tilde{I}$;
5. T' contains at most one definition;
6. there is a one-to-one correspondence between the models of T approximated by $\tilde{I}$ and the models of T' expanding J.

Conditions 1, 2 and 3 are obtained as described below the definitions of TNF (Definition 2.1 and 3.14): first all negations are moved inside, next function symbols with a three-valued interpretation are moved outside by applying equivalences (2.13) and (2.14), and finally, aggregate expressions are rewritten to the desired format as described below Definition 3.14. Denote the resulting theory by T_1.
The fourth condition is obtained by calculating known definitions (see Section 5.5.1): for every definition Δ in T_1 such that $\tilde{I}$ is two-valued on $\mathrm{Open}(\Delta)$, the well-founded model of Δ extending $\tilde{I}$ is computed, the interpretation of $P \in \mathrm{Def}(\Delta)$ is set to $P^{\mathrm{wfm}_\Delta(\tilde{I})}$ and Δ is omitted from T_1. This process is repeated until every definition in the theory depends on predicates with a strictly three-valued interpretation in $\tilde{I}$. Denote the resulting theory by T_2 and the resulting structure by $\tilde{I}_2$. If a predicate P is defined in multiple definitions whose well-founded model is computed during this rewriting phase, it is checked if the different calculated interpretations of P coincide. If this is not the case, GIDL reports immediately that T has no model approximated by $\tilde{I}$. Similarly if one of the well-founded models turns out to be strictly three-valued.
The fifth condition is obtained by merging the definitions (Section 3.5.3) of T_2 in the following manner. If a predicate P is defined by definitions $\Delta_1, \ldots, \Delta_n$ of T, new predicates $P_2, \ldots, P_n$ are introduced, every occurrence of P in Δ_i, $2 \leq i \leq n$, is replaced by P_i, and the sentences $\forall \bar{x}\ (P(\bar{x}) \Leftrightarrow P_i(\bar{x}))$ are added to T_2. Similarly if P has a two-valued interpretation in $\tilde{I}_2$ and is defined by a definition Δ in T_2: in this case the occurrences of P in Δ are replaced by a new symbol P' and the sentence $\forall \bar{x}\ (P(\bar{x}) \Leftrightarrow P'(\bar{x}))$ is added. Denote the resulting theory by T_3.
Consider the graph $G = \langle V, E \rangle$ where V is the set of atoms that are defined by at least (and therefore by exactly) one definition in T_3 and E contains edge

(P, Q) if P occurs in the body of a rule defining Q. If G contains a cycle C such that not all predicates in C are defined by the same definition in T_3, one predicate P of C is chosen. Let Δ be the definition of T_3 such that $P \in \mathrm{Def}(\Delta)$. Then a new predicate P' is introduced, all occurrences of P in Δ are replaced by P' and the sentence $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow P'(\overline{x}))$ is added. This process is repeated until for every cycle C in the constructed graph, all predicates of C are defined by the same definition. Denote by T_4 the resulting theory. Then the theory T_5, obtained from T_4 by replacing all definitions in T_4 by a single definition consisting of all rules that occur in T_4, is Σ-equivalent to T_4. Hence there is a one-to-one correspondence between the models of T approximated by $\tilde{I}$ and the models of T_5 approximated by $\tilde{I}_2$.
Finally, to obtain the two-valued structure J and the theory T', let σ be the vocabulary that contains all symbols that have a two-valued interpretation in $\tilde{I}_2$ and two symbols P^{ct} and P^{cf} for every symbol P with a three-valued interpretation in $\tilde{I}$. Define $P^J = P^{\tilde{I}}$ for every symbol P such that $P^{\tilde{I}}$ is two-valued and define $(P^{\mathrm{ct}})^J = P^{\tilde{I}^{\mathrm{ct}}}$ and $(P^{\mathrm{cf}})^J = P^{\tilde{I}^{\mathrm{cf}}}$ for every symbol $P \in \Sigma$ such that $P^{\tilde{I}}$ is strictly three-valued. T' is obtained from T_5 by adding the sentences $\forall \overline{x}\ (P^{\mathrm{ct}}(\overline{x}) \Rightarrow P(\overline{x}))$ and $\forall \overline{x}\ (P^{\mathrm{cf}}(\overline{x}) \Rightarrow \neg P(\overline{x}))$ for every P such that $P^{\tilde{I}}$ is strictly three-valued.

Symbolic propagation

The symbolic propagation algorithm implemented in GIDL to compute c-maps is basically the one described in Section 4.3.3. Below, we discuss the differences and present the estimators that implement the stop criterion of the symbolic propagation algorithm in GIDL. We also mention some simplification strategies for BDDs.

Propagators in GIDL

The first difference in GIDL compared to the symbolic propagation method of Chapter 4 is that for an ENF sentence of the form $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \forall y\ L[\overline{x}, y])$, not all INF propagators are used. Specifically, the implementation of the propagation method in GIDL does not use the INF propagator associated to $\forall \overline{x} \forall y\ (\neg P(\overline{x}) \wedge (\forall y'\ (y \neq y' \Rightarrow L[\overline{x}, y'])) \Rightarrow \neg L[\overline{x}, y])$. The rationale behind this decision is the fact that applying this INF propagator will very often lead to a bound that is too complex to be useful for grounding. Indeed, the formula $\neg P(\overline{x}) \wedge (\forall y'\ (y \neq y' \Rightarrow L[\overline{x}, y']))$ can only be true in a structure $\tilde{I}$ in the (rare) case where there exists a $\overline{d}$ such that $L[\overline{d}, d']$ is true in $\tilde{I}$ for all but one d' in the domain of $\tilde{I}$. Even if the propagator would be applied, the computed formula would almost never pass the check `acceptable` in Algorithm 4.3. For exactly the same reason, for the ENF sentence $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow \exists y\ L[\overline{x}, y])$, the INF propagator associated to $\forall \overline{x} \forall y\ (P(\overline{x}) \wedge (\forall y'\ (y \neq y' \Rightarrow \neg L[\overline{x}, y'])) \Rightarrow L[\overline{x}, y])$ is not applied. Also, we did not yet implement propagators for ENF sentences involving aggregates.

Secondly, to implement propagation for a sentence φ of the form $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow L_1[\overline{y}_1] \wedge \ldots \wedge L_n[\overline{y}_n])$ where $n > 2$, φ is not rewritten to a set of ENF sentences. Instead, for every $1 \leq i \leq n$ the following INF sentences are associated to φ and used for propagation:

$$\begin{aligned}
&\forall \overline{x}\ (L_1[\overline{y}_1] \wedge \ldots \wedge L_n[\overline{y}_n] \Rightarrow P(\overline{x})),\\
&\forall \overline{x}\ (\neg L_i[\overline{y}_i] \Rightarrow \neg P(\overline{x})),\\
&\forall \overline{y}_i\ ((\exists \overline{z}_i\ P(\overline{x})) \Rightarrow L_i[\overline{y}_i]),\\
&\forall \overline{y}_i\ ((\exists \overline{z}_i\ \neg P(\overline{x}) \wedge L_1[\overline{y}_1] \wedge \ldots \wedge L_{i-1}[\overline{y}_{i-1}]\\
&\qquad \wedge L_{i+1}[\overline{y}_{i+1}] \wedge \ldots \wedge L_n[\overline{y}_n]) \Rightarrow \neg L_i[\overline{y}_i]).
\end{aligned}$$

Here, $\overline{z}_i$ are the variables in $\overline{x}$ that do not occur in $\overline{y}_i$. Observe that this is a generalization of the case where $n = 2$ (see Table 4.1). A similar strategy is applied if φ is of the form $\forall \overline{x}\ (P(\overline{x}) \Leftrightarrow L_1[\overline{y}_1] \vee \ldots \vee L_n[\overline{y}_n])$

Simplifying BDDs

Besides the simplification techniques for BDDs presented by Goubault (1995), GIDL implements a few simplification techniques for reasoning with equality. Specifically, in GIDL

- a BDD of the form $x = x \rightarrowtriangle \psi_1; \psi_2$ is simplified to ψ_1;
- a BDD of the form $(\exists x\ F(\overline{y}) = x) \rightarrowtriangle \psi_1; \psi_2$ is simplified to ψ_1 if F is a total function with sorts $(s_1, \ldots, s_{n+1})$ and $\mathfrak{s}(x) = s_{n+1}$;
- a BDD of the form $s_{\text{pred}}(x) \rightarrowtriangle \psi_1; \psi_2$ is replaced by ψ_1 if $\mathfrak{s}(x) = s$;
- a BDD of the form $x = y \rightarrowtriangle \psi_1; \psi_2$ is replaced by $x = y \rightarrowtriangle \psi_1[y/x]; \psi_2$.

We also implemented a form of simplification by partial evaluation (see Section 4.3.3), namely replacing a BDD φ without free variables by $\top$ if it satisfied in the input structure, and by $\bot$ otherwise. However, experiments showed that this form of simplification by partial evaluation does not lead to faster grounding times.

Estimators

In Section 5.4.2 we explained how estimators for the number of answers to a bound and for the cost of querying a bound can be used to implement the stop criterion for the symbolic propagation phase in a grounder. In this section, we provide details on the estimators implemented in GIDL.

For the rest of this section, we assume a fixed vocabulary Σ and a fixed finite Σ-structure I. For a tuple $\overline{x}$ of variables, we denote by $\|\overline{x}\|$ the cardinality of $\mathfrak{s}(\overline{x})^I$. If $\overline{x}$ is the empty tuple, then $\|\overline{x}\| = 1$. For a predicate P, $\|P\|$ denotes the cardinality of P^I. To simplify the presentation, we assume Σ does not contain function symbols.

Number of answers

The first estimator we present estimates the number of answers to a query $\{\overline{x} \mid \varphi\}$ in I, where φ is a BDD or kernel over Σ with free variables among $\overline{x}$. The estimated number of answers is denoted by $Ans(\varphi, \overline{x})$ and defined by structural induction:

- $Ans(\bot, \overline{x}) = 0$;
- $Ans(\top, \overline{x}) = \|\overline{x}\|$;
- For a BDD φ of the form $\psi[\overline{y}] \multimap \psi_1; \psi_2$,

$$\begin{aligned} Ans(\varphi, \overline{x}) = &\big(Ans(\psi, \overline{y}) \cdot Ans(\psi_1[\overline{y}/\overline{d}], \overline{x} \setminus \overline{y})\big) \\ &+ \big((\|\overline{y}\| - Ans(\psi, \overline{y})) \cdot Ans(\psi_2[\overline{y}/\overline{d}], \overline{x} \setminus \overline{y})\big)\,, \end{aligned}$$

  where $\overline{d}$ is an arbitrary tuple of domain elements in $\mathfrak{s}(\overline{y})^I$. The rationale behind this formula is as follows: an answer to φ is either an answer to ψ and ψ_1 or an answer to $\neg\psi$ and ψ_2. The number of answers that correspond to the former case is given by $\big(Ans(\psi, \overline{y}) \cdot Ans(\psi_1[\overline{y}/\overline{d}], \overline{x} \setminus \overline{y})\big)$, i.e., by the estimated number of answers $\overline{d}$ to ψ multiplied by the estimated number of answers to ψ_1 given that $\overline{y}$ is interpreted by $\overline{d}$. Similarly, the number of answers that correspond to the latter case is estimated by $\big((\|\overline{y}\| - Ans(\psi, \overline{y})) \cdot Ans(\psi_2[\overline{y}/\overline{d}], \overline{x} \setminus \overline{y})\big)$.

- For a kernel $P(\bar{t})$, the estimated number of answers is defined by the probability a tuple $\overline{d}$ belongs to P^I, multiplied by the number of tuples in $\mathfrak{s}(\overline{x})^I$. That is,

$$Ans(P(\bar{t}), \overline{x}) = \frac{\|P\|}{|\mathfrak{s}(P)^I|} \cdot \|\overline{x}\|.$$

- For a kernel φ of the form $\exists z\ \psi[\overline{y}, z]$, the estimated probability that $(\overline{d}_y, d_z)$ is an answer to $\{(\overline{y}, z) \mid \psi\}$ in I is given by $Ans(\psi, (\overline{y}, z))/\|(\overline{y}, z)\|$. Hence, the estimated probability that $\overline{d}_y$ is *not* an answer to $\{\overline{y} \mid \varphi\}$ is given by

$$\left(1 - \frac{Ans(\psi, (\overline{y}, z))}{\|(\overline{y}, z)\|}\right)^{\|z\|}.$$

  We can now define the estimated number of answers to φ by the probability that a tuple is an answer to φ, multiplied by the number of tuples in $\mathfrak{s}(\overline{x})^I$. That is,

$$Ans(\varphi, \overline{x}) = \left(1 - \left(1 - \frac{Ans(\psi, (\overline{y}, z))}{\|(\overline{y}, z)\|}\right)^{\|z\|}\right) \cdot \|\overline{x}\|.$$

Example C.1 (Example 5.8 ctd.)**.** In the clique problem presented in Example 5.8 the ct-bound $\forall x'\ (x \neq x' \Rightarrow \mathit{Edge}(x, x'))$ was derived for formula

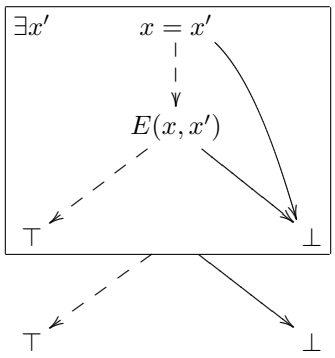


Figure C.1: A BDD representing the ct-bound for *Clique*(x)

Clique(x). A BDD representation of this bound is given in Figure C.1 Denote this BDD by φ, denote by χ the subtree *Edge(x,x')* $\rightarrowtriangle \bot; \top$ of φ and denote by ψ the subtree $x = x' \rightarrowtriangle \bot; \chi$ of φ. If I is a structure with domain D, then according to the definition of Ans,

$$\begin{aligned} Ans(\chi[x/d, x'/d'], \emptyset) &= (Ans(Edge(d, d'), \emptyset) \cdot Ans(\bot, \emptyset)) \\ &\quad + ((1 - Ans(Edge(d,d'), \emptyset)) \cdot Ans(\top, \emptyset)) \\ &= 1 - Ans(Edge(d,d'), \emptyset) \\ &= (1 - \|Edge\|/D^2), \end{aligned}$$

$$\begin{aligned} Ans(\psi, (x, x')) &= (\|(x, x')\| - Ans(x = x', (x, x'))) \cdot Ans(\chi[x/d, x'/d'], \emptyset) \\ &= (D^2 - D) \cdot (1 - \|Edge\|/D^2) \end{aligned}$$

and

$$\begin{aligned} Ans(\varphi, x) &= \|x\| - Ans(\exists x'\ \psi, x) \\ &= D - \left(1 - \left(1 - \frac{Ans(\psi, (x, x'))}{\|(x, x')\|}\right)^{\|x'\|}\right) \cdot \|x\| \\ &= D - \left(1 - \left(1 - \frac{(D^2 - D) \cdot (1 - \|Edge\|/D^2)}{D^2}\right)^{D}\right) \cdot D. \end{aligned}$$

One can check that for a graph with 10 nodes, i.e., $|D| = 10$, $Ans(\varphi, x) > 1$ if there are more than 77 edges in the graph. For a graph with 100 nodes, more than 9545 edges are required before $Ans(\varphi, x) > 1$. We conclude that only for very dense graphs there is a chance that the ct-bound φ has an answer.

Cost of computing answers

We define two functions to estimate the cost of computing the answers to a query. For a a BDD, kernel or negated kernel $\varphi[\overline{y}]$ and a tuple of variables $\overline{x} \supseteq \overline{y}$, $Cost_{one}(\varphi, \overline{x})$ estimates the average cost of computing one answer to the query $\{\overline{x} \mid \varphi\}$, $Cost_{all}(\varphi, \overline{x})$ the cost of computing all answers to that query. The definitions of $Cost_{one}(\varphi, \overline{x})$ and $Cost_{all}(\varphi, \overline{x})$ are given by structural induction and depend on the data-structures in GIDL:

- $Cost_{all}(\top, \overline{x}) = \|\overline{x}\|$, i.e., we (arbitrarily) define that the cost of simply returning n answers, is n.
- $Cost_{all}(\bot, \overline{x}) = 1$, i.e., we define the cost of checking that there are no answers to $\bot$ by 1.
- Let φ be a kernel of the form $P(t_1, \ldots, t_n)$ where each of the t_i is either a variable or a domain constant. Let $t_{m_1}, \ldots, t_{m_k}$ be the terms among $(t_1, \ldots, t_n)$ that are domain constants. If it is detected that a query of the form $\{\overline{x} \mid \varphi\}$ needs to be answered during grounding, GIDL stores a function

  $$F : D^k \to \mathscr{P}(D^{n-k}) : (d_1, \ldots, d_k) \mapsto \{\overline{x} \mid \varphi[t_{m_1}/d_1, \ldots, t_{m_k}/d_k]\}^I.$$

  The function F is implemented in GIDL as a sorted associative array, allowing to compute a value $F(d_1, \ldots, d_m)$ in time logarithmic in the number of tuples $\overline{d}$ such that $F(\overline{d}) \neq \emptyset$.[33] The values in $F(d_1, \ldots, d_m)$ are themselves stored in a sorted array. Also, the array implementing F is computed only once: all queries of the form $\{\overline{y} \mid P(t'_1, \ldots, t'_n)\}$, such that $t'_{m_1}, \ldots, t'_{m_k}$ are domain constants are answered using the same array. As such, we do not take the time needed to construct this array into account and we define

  $$\begin{aligned} Cost_{all}(\varphi, \overline{x}) = (1 + \log_2(Ans(\exists \overline{x}\ \varphi[t_{m_1}/z_1, \ldots, t_{m_k}/z_k], (z_1, \ldots, z_k)))) \\ + Ans(\varphi, \overline{x}). \end{aligned}$$

  To compute all answers to the query $\{\overline{x} \mid \psi\}$ where ψ is the negated kernel $\neg\varphi$, first $F(t_{m_1}, \ldots, t_{m_k})$ is computed. Next, for each tuple $\overline{d} \in \mathfrak{s}(\overline{x})^I$ it is checked whether $\overline{d} \in F(t_{m_1}, \ldots, t_{m_k})$. Since the set $F(t_{m_1}, \ldots, t_{m_k})$ is implemented by a sorted array, this takes time $\mathcal{O}(\|\overline{x}\|)$. Therefore we define

  $$\begin{aligned} Cost_{all}(\psi, \overline{x}) = (1 + \log_2(Ans(\exists \overline{x}\ \varphi[t_{m_1}/z_1, \ldots, t_{m_k}/z_k], (z_1, \ldots, z_k)))) \\ + \|\overline{x}\|. \end{aligned}$$

- Now let φ be the kernel $\exists z\ \psi[\overline{y}, z]$. To compute answers for the query $\{\overline{x} \mid \varphi\}$, first an answer $(\overline{d}_y, d_z)$ for $\{(\overline{y}, z) \mid \psi\}$ is computed. Then for every $\overline{d}' \in \mathfrak{s}(\overline{x} \setminus \overline{y})^I$, $(\overline{d}', \overline{d}_y)$ is an answer to $\{\overline{x} \mid \varphi\}$. The next set of answers

[33] The performance of GIDL could be improved by implementing F by a hash table instead.

to $\{\overline{x} \mid \varphi\}$ is obtained by computing an answer $(\overline{d}'_y, d'_z)$ for $\{(\overline{y}, z) \mid \psi\}$ such that $\overline{d}'_y \neq \overline{d}_y$. Therefore, we define the cost to compute all answers to $\{\overline{x} \mid \varphi\}$ by

$$Cost_{all}(\varphi, \overline{x}) = (Cost_{one}(\psi, (\overline{y}, z)) + \|\overline{x} \setminus \overline{y}\|) \cdot Ans(\varphi, \overline{y}).$$

- If φ is the negated kernel $\neg(\exists z\ \psi[\overline{y}, z])$, then answers to $\{\overline{x} \mid \varphi\}$ are computed by guessing a tuple $\overline{d}_y$, checking whether $\psi[\overline{y}/\overline{d}_y, z]$ has an answer and if this is not the case, returning tuples of the form $(\overline{d}', \overline{d}_y)$ where $\overline{d}' \in \mathfrak{s}(\overline{x} \setminus \overline{y})^I$ as answers to $\{\overline{x} \mid \varphi\}$. This suggests to define

$$Cost_{all}(\varphi, \overline{x}) = (\|\overline{y}\| \cdot Cost_{one}(\psi[\overline{y}/\overline{d}, z], z)) + (\|\overline{y}\| - Ans(\varphi, \overline{y})) \cdot \|\overline{x} \setminus \overline{y}\|.$$

- If φ is the BDD $\psi[\overline{z}] \twoheadrightarrow \psi_1; \bot$,

$$Cost_{all}(\varphi, \overline{x}) = Cost_{all}(\psi, \overline{z}) + Ans(\psi, \overline{z}) \cdot Cost_{all}(\psi_1, \overline{x} \setminus \overline{z}).$$

- If φ is the BDD $\psi[\overline{z}] \twoheadrightarrow \bot; \psi_2$,

$$Cost_{all}(\varphi, \overline{x}) = Cost_{all}(\neg\psi, \overline{z}) + (\|\overline{z}\| - Ans(\psi, \overline{z})) \cdot Cost_{all}(\psi_2, \overline{x} \setminus \overline{z}).$$

- If φ is the BDD $\psi[\overline{z}] \twoheadrightarrow \psi_1; \psi_2$ and ψ_1 nor ψ_2 are equal to $\bot$, then

$$\begin{aligned} Cost_{all}(\varphi, \overline{x}) = &\|\overline{z}\| \cdot Cost_{all}(\psi[\overline{z}/\overline{d}], \emptyset) \\ &+ Ans(\psi, \overline{z}) \cdot Cost_{all}(\psi_1, \overline{x} \setminus \overline{z}) \\ &+ (\|\overline{z}\| - Ans(\psi, \overline{z})) \cdot Cost_{all}(\psi_2, \overline{x} \setminus \overline{z}) \end{aligned}$$

- For any BDD, kernel or negated kernel φ,

$$Cost_{one}(\varphi, \overline{x}) = Cost_{all}(\varphi, \overline{x}) / \max(1, Ans(\varphi, \overline{x})).$$

Example C.2 (Example C.1 ctd.)**.** Let φ be the BDD in Figure C.1 and I a structure with domain size 10. In Example C.1 it was shown that more than 77 edges are needed before the estimated number of answers to $\{x \mid \varphi\}$ is more than one. On the other hand, one can check that if $\|Edge\| > 77$, then $Cost_{all}(\varphi, x) > 292$. This indicates that φ is probably not a useful bound: it is estimated that it has few answers, which are, moreover, very expensive to compute.

In GIDL, a bound $\varphi[\overline{x}]$ is not used to replace a bound $\psi[\overline{x}]$ if

$$\frac{Ans(\varphi, \overline{x}) + 1}{(Cost_{one}(\varphi, \overline{x}))^C + 1)} \leq \frac{Ans(\psi, \overline{x}) + 1}{(Cost_{one}(\psi, \overline{x}))^C + 1)}.$$

Here C is a given constant, which was set to 2 in the experiments reported on in Section 5.4.

Translation file

The output of GIDL for an input theory T, structure I_σ and bounds $\mathcal{C}$ consists of two files. The first file, called the *translation file* defines an injective function Num from the domain atoms over expansion symbols to natural numbers and contains the grounding of $\overline{\mathcal{C}}_A$. The second file, the *grounding file* contains $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle, I_\sigma)$, where each domain atom $P(\overline{d})$ is replaced by $\mathrm{Num}(P(\overline{d}))$.
The function Num is defined by assigning an offset N_P to every predicate symbol P. For a domain atom $P(\overline{d})$, $\mathrm{Num}(P(\overline{d}))$ is then the number $N_P + n$, where n is the position of $\overline{d}$ in the lexicographic ordering of the set $\mathfrak{s}(P)^{I_\sigma}$. Similarly for function symbols. The offsets are chosen such that Num is bijective on an interval $[1, N]$ of the natural numbers. The translation file contains for every sort s its domain s^{I_σ} and for every expansion symbol its sorts and offset, which is enough information to compute Num. There are two benefits of this approach, compared to storing an exhaustive table linking each domain element to a unique number. First, it allows for small translation files. Secondly, $\mathrm{Num}(P(\overline{d}))$ and $\mathrm{Num}^{-1}(n)$ can be computed in constant time in the number of domain atoms. On the other hand, the interval $[1, N]$ can be much larger than necessary since it can be the case that only few of the numbers in $[1, N]$ actually occur in the grounding file. This may slow down SAT solvers severely since it causes them to store an unnecessary large amount of data.
The translation file contains a list of all (numbers associated to) atomic sentences that occur in the grounding of $\overline{\mathcal{C}}_A$. Furthermore, it contains a list *Arbit* of all domain atoms $P(\overline{d})$ such that neither $P(\overline{d})$ nor $\neg P(\overline{d})$ is a sentence of $\overline{\mathcal{C}}_A$ and $\mathrm{Num}(P(\overline{d}))$ does not occur in the ground file. For every $P(\overline{d}) \in$ *Arbit* and for every structure M such that $M \models_{I_\sigma} T$, also $\mathrm{swap}(M, P(\overline{d})) \models_{I_\sigma} T$. That is, the value of domain atoms in *Arbit* can be chosen arbitrarily in solutions to the model expansion problem with input $\langle T, I_\sigma\rangle$. To compute the list *Arbit*, GIDL stores the numbers of the domain atoms that occur in the ground file. Then, it checks for every domain atom $P(\overline{d})$ that does not occur in the ground file whether $\overline{d} \in \{\overline{x} \mid \mathcal{C}^{\mathrm{ctb}}(P(\overline{x})) \vee \mathcal{C}^{\mathrm{cfb}}(P(\overline{x}))\}^{I_\sigma}$. If this is not the case, $P(\overline{d})$ is added to *Arbit*. Observe that $\neg P(\overline{d})$ occurs in the grounding of $\overline{\mathcal{C}}_A$ iff $P(\overline{d})$ is not an atomic sentence in the grounding of $\overline{\mathcal{C}}_A$ and does not occur in *Arbit* or in the ground file. Hence the grounding of $\overline{\mathcal{C}}_A$ can be reconstructed from the ground and translation files. The benefit of this implicit representation of $\overline{\mathcal{C}}_A$ is that for most practical model expansion problems, the list *Arbit* is much smaller than the number of negative literals in the grounding of $\overline{\mathcal{C}}_A$.

Bibliography

Arieli, O., Denecker, M., Nuffelen, B. V., and Bruynooghe, M. (2004). Coherent integration of databases by abductive logic programming. *Journal of Artificial Intelligence Research*, 21:245–286.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baral, C., Brewka, G., and Schlipf, J. S., editors (2007). *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*. Springer.

Bertossi, L. E. and Schwind, C. (2004). Database repairs and analytic tableaux. *Annals of Mathematics and Artificial Intelligence*, 40(1-2):5–35.

Brain, M., Watson, R., and Vos, M. D. (2005). An interactive approach to answer set programming. In Vos, M. D. and Provetti, A., editors, *International Workshop on Answer Set Programming (ASP)*, volume 142 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Brewka, G. (2004). Answer sets: From constraint programming towards qualitative optimization. In Lifschitz, V. and Niemelä, I., editors, *International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 2923 of *Lecture Notes in Computer Science*, pages 34–46. Springer.

Brewka, G. and Lang, J., editors (2008). *Principles of Knowledge Representation and Reasoning: Proceedings of the 11th International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*. AAAI Press.

Bruynooghe, M. (1991). A practical framework for the abstract interpretation of logic programs. *Journal of Logic Programming*, 10(1/2/3&4):91–124.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691.

Buchholz, W., Feferman, S., Pohlers, W., and Sieg, W. (1981). *Iterated Inductive Definitions and Subsystems of Analysis : Recent Proof-Theoretical Studies*, volume 897 of *Lecture Notes is Mathematics*. Springer.

Cadoli, M., Ianni, G., Palopoli, L., Schaerf, A., and Vasile, D. (2000). NP-SPEC: an executable specification language for solving all problems in NP. *Computer Languages*, 26(2-4):165–195.

Cadoli, M. and Schaerf, A. (2005). Compiling problem specifications into SAT. *Artificial Intelligence*, 162(1-2):89–120.

Catalano, G., Leone, N., and Perri, S. (2008). On demand indexing for the DLV instantiator. In Faber, W. and Lee, J., editors, *Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*.

Chang, F. (2007). Alloy analyzer 4.0. `http://alloy.mit.edu/alloy4/`.

Claessen, K. and Sörensson, N. (2003). New techniques that improve MACE-style model finding. In *Workshop on Model Computation (MODEL)*.

Cliffe, O., De Vos, M., Brain, M., and Padget, J. A. (2008). ASPVIZ: Declarative visualisation and animation using answer set programming. In García de la Banda and Pontelli (2008), pages 724–728.

Cortés Calabuig, A. (2008). *Towards a logical reconstruction of a theory for locally complete databases*. PhD thesis, K.U.Leuven, Leuven, Belgium.

Cortés Calabuig, Á., Denecker, M., Arieli, O., and Bruynooghe, M. (2006). Representation of partial knowledge and query answering in locally complete databases. In Hermann, M. and Voronkov, A., editors, *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 4246 of *Lecture Notes in Computer Science*, pages 407–421. Springer.

Cortés Calabuig, Á., Denecker, M., Arieli, O., and Bruynooghe, M. (2007). Approximate query answering in locally closed databases. In Holte and Howe (2007), pages 397–402.

Cortés-Calabuig, Á., Denecker, M., Arieli, O., and Bruynooghe, M. (2008). Accuracy and efficiency of fixpoint methods for approximate query answering in locally complete databases. In Brewka and Lang (2008), pages 81–91.

D'Agostino, M., Gabbay, D. M., Hähnle, R., and Posegga, J. (1999). *Handbook of Tableau Methods*. Kluwer Academic Publishers, Dordrecht.

Davis, M., Logemann, G., and Loveland, D. W. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215.

De Cat, B. (2009). Development of algorithms for modelrevision with applications in trainscheduling and networkconfiguration. Master's thesis, Katholieke Universiteit Leuven, Leuven, Belgium. in Dutch.

Demolombe, R. (1980). Estimation of the number of tuples satisfying a query expressed in predicate calculus language. In *International Conference on Very Large Data Bases (VLDB)*, pages 55–63. IEEE Computer Society.

Denecker, M. (2000). Extending classical logic with inductive definitions. In Lloyd, J. W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Palamidessi, C., Pereira, L. M., Sagiv, Y., and Stuckey, P. J., editors, *International Conference on Computational Logic (CL)*, volume 1861 of *Lecture Notes in Computer Science*, pages 703–717. Springer.

Denecker, M. and Ternovska, E. (2007). Inductive situation calculus. *Artificial Intelligence*, 171(5-6):332–360.

Denecker, M. and Ternovska, E. (2008). A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic (TOCL)*, 9(2):Article 14.

Denecker, M. and Vennekens, J. (2007). Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In Baral et al. (2007), pages 84–96.

Denecker, M. and Vennekens, J. (2008). Building a knowledge base system for an integration of logic programming and classical logic. In García de la Banda and Pontelli (2008), pages 71–76.

Denecker, M., Vennekens, J., Bond, S., Gebser, M., and Truszczyński, M. (2009). The second answer set programming competition. In Erdem et al. (2009), pages 637–654.

Ducassé, M. (1999). Opium: An extendable trace analyzer for Prolog. *Journal of Logic Programming*, 39(1-3):177–223.

East, D., Iakhiaev, M., Mikitiuk, A., and Truszczyński, M. (2006). Tools for modeling and solving search problems. *AI Communications*, 19(4):301–312.

East, D. and Truszczyński, M. (2006). Predicate-calculus-based logics for modeling and solving search problems. *ACM Transactions on Computational Logic (TOCL)*, 7(1):38–83.

Enderton, H. B. (1977). *Elements of Set Theory*. Academic Press.

Erdem, E., Lin, F., and Schaub, T., editors (2009). *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, volume 5753 of *Lecture Notes in Computer Science*. Springer.

Etalle, S. and Truszczyński, M., editors (2006). *22nd International Conference on Logic Programming, ICLP 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4079 of *Lecture Notes in Computer Science*. Springer.

Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T., and Ielpa, G. (2008). Design and implementation of aggregate functions in the DLV system. *Theory and Practice of Logic Programming (TPLP)*, 8(5-6):545–580.

Fagin, R. (1974). Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of Computation*, 7:43–74.

Faustino da Silva, A. and Santos Costa, V. (2006). The design of the YAP compiler: An optimizing compiler for logic programming languages. *Journal of Universal Computer Science*, 12(7):764–787.

Forgy, C. (1982). Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37.

Fox, D. and Gomes, C. P., editors (2008). *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. AAAI Press.

Fox, M., Gerevini, A., Long, D., and Serina, I. (2006). Plan stability: Replanning versus plan repair. In Long, D., Smith, S. F., Borrajo, D., and McCluskey, L., editors, *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 212–221. AAAI.

Frege, G. (1892). Über sinn und bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, NF(100):25–50.

García de la Banda, M. and Pontelli, E., editors (2008). *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, volume 5366 of *Lecture Notes in Computer Science*. Springer.

Garcia-Molina, H., Ullman, J. D., and Widom, J. (2000). *Database System Implementation*. Prentice-Hall.

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Gebser, M., Kaminski, R., Ostrowski, M., Schaub, T., and Thiele, S. (2009a). On the input language of ASP grounder Gringo. In Erdem et al. (2009), pages 502–508.

Gebser, M., Ostrowski, M., and Schaub, T. (2009b). Constraint answer set solving. In Hill and Warren (2009), pages 235–249.

Gebser, M., Pührer, J., Schaub, T., and Tompits, H. (2008). A meta-programming technique for debugging answer-set programs. In Fox and Gomes (2008), pages 448–453.

Gebser, M. and Schaub, T. (2006). Tableau calculi for answer set programming. In Etalle and Truszczyński (2006), pages 11–25.

Gebser, M., Schaub, T., and Thiele, S. (2007). GrinGo : A new grounder for answer set programming. In Baral et al. (2007), pages 266–271.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. A. and Bowen, K. A., editors, *International Conference on Logic Programming (ICLP/SLP)*, pages 1070–1080. MIT Press.

Goubault, J. (1995). A BDD-based simplification and skolemization procedure. *Logic Journal of IGPL*, 3(6):827–855.

Grädel, E., Kolaitis, P. G., Libkin, L., Marx, M., Spencer, J., Vardi, M. Y., Venema, Y., and Weistein, S. (2007). *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. Springer.

Grahne, G. and Mendelzon, A. O. (1999). Tableau techniques for querying information sources through global schemas. In Beeri, C. and Buneman, P., editors, *International Conference on Database Theory (ICDT)*, volume 1540 of *Lecture Notes in Computer Science*, pages 332–347. Springer.

Hill, P. M. and Warren, D. S., editors (2009). *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, volume 5649 of *Lecture Notes in Computer Science*. Springer.

Holte, R. C. and Howe, A., editors (2007). *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*. AAAI Press.

Hou, P. (2010). *Techniques for Reasoning in FO(ID) and Fixpoint Logic*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium.

Hou, P. and Denecker, M. (2009). A logic of fixpoint definitions. In Faber, W. and Lee, J., editors, *Second Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*.

Jackson, D. (2006). *Software Abstractions: Logic, Language, and Analysis*. MIT Press, Cambridge, MA.

Jahier, E., Ducassé, M., and Ridoux, O. (2000). Specifying Prolog trace models with a continuation semantics. In Lau, K.-K., editor, *International Workshop on Logic Based Program Synthesis and Transformation (LOPSTR)*, volume 2042 of *Lecture Notes in Computer Science*, pages 165–182. Springer.

Kautz, H. A. and Selman, B. (1996). Pushing the envelope: Planning, propositional logic and stochastic search. In *National Conference on Artificial Intelligence and Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 1194–1201. AAAI Press.

Kleene, S. C. (1952). *Introduction to Metamathematics*. Van Nostrand.

Krogel, M.-A., Rawles, S., Železný, F., Flach, P. A., Lavrac, N., and Wrobel, S. (2003). Comparative evaluation of approaches to propositionalization. In Horváth, T., editor, *International Conference on Inductive Logic Programming (ILP)*, volume 2835 of *Lecture Notes in Computer Science*, pages 197–214. Springer.

Langevine, L., Ducassé, M., and Deransart, P. (2003). A propagation tracer for GNU-Prolog: From formal definition to efficient implementation. In Palamidessi, C., editor, *International Conference on Logic Programming (ICLP)*, volume 2916 of *Lecture Notes in Computer Science*, pages 269–283. Springer.

Leone, N., Perri, S., and Scarcello, F. (2001). Improving ASP instantiators by join-ordering methods. In Eiter, T., Faber, W., and Truszczyński, M., editors, *International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 2173 of *Lecture Notes in Computer Science*, pages 280–294. Springer.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562.

Leuschel, M. (1997). Advanced techniques for logic program specialisation. *AI Communications*, 10(2):127–128.

Li, C. M. and Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 366–371. Morgan Kaufman.

Li, C. M., Manyà, F., and Planes, J. (2007). New inference rules for max-sat. *Journal of Artificial Intelligence Research (JAIR)*, 30:321–359.

Mallet, S. and Ducassé, M. (1999). Generating deductive database explanations. In De Schreye, D., editor, *International Conference on Logic Programming (ICLP)*, pages 154–168. MIT Press.

Marek, V. W. and Truszczyński, M. (1999). Stable models and an alternative logic programming paradigm. In Apt, K. R., Marek, V. W., Truszczyński, M., and Warren, D. S., editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag.

Mariën, M. (2009). *Model Generation for ID-Logic*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium.

Mariën, M., Gilis, D., and Denecker, M. (2004). On the relation between ID-Logic and Answer Set Programming. In Alferes, J. J. and Leite, J. A., editors, *European Conference on Logics in Artificial Intelligence (JELIA)*, volume 3229 of *Lecture Notes in Computer Science*, pages 108–120. Springer.

Mariën, M., Wittocx, J., and Denecker, M. (2006). The IDP framework for declarative problem solving. In *Search and Logic: Answer Set Programming and SAT*, pages 19–34.

Mariën, M., Wittocx, J., and Denecker, M. (2007a). Integrating inductive definitions in SAT. In Derschowitz, N. and Voronkov, A., editors, *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 4790 of *Lecture Notes in Computer Science*, pages 378–392. Springer.

Mariën, M., Wittocx, J., and Denecker, M. (2007b). MidL: a SAT(ID) solver. In *4th Workshop on Answer Set Programming: Advances in Theory and Implementation*, pages 303–308.

Mariën, M., Wittocx, J., Denecker, M., and Bruynooghe, M. (2008). SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In Kleine Büning, H. and Zhao, X., editors, *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4996 of *Lecture Notes in Computer Science*, pages 211–224. Springer.

McCune, W. (2003). Mace4 reference manual and guide. *CoRR*, cs.SC/0310055.

Meier, M. (1995). Debugging constraint programs. In Montanari, U. and Rossi, F., editors, *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 976 of *LNCS*, pages 204–221. Springer.

Mellarkod, V. S., Gelfond, M., and Zhang, Y. (2008). Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):251–287.

Michel, L. and Van Hentenryck, P. (2005). The Comet programming language and system. In van Beek, P., editor, *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3709 of *Lecture Notes in Computer Science*, pages 881–881. Springer.

Miranker, D. P., Brant, D. A., Lofaso, B. J., and Gadbois, D. (1990). On the performance of lazy matching in production systems. In Dieterich, T. and Swartout, W., editors, *National Conference on Artificial Intelligence (AAAI)*, pages 685–692. AAAI/MIT Press.

Mitchell, D. G. (2005). A SAT solver primer. *Bulletin of the European Association for Theoretical Computer Science*, 85:112–132.

Mitchell, D. G. and Ternovska, E. (2005). A framework for representing and solving NP search problems. In Veloso and Kambhampati (2005), pages 430–435.

Mitchell, D. G. and Ternovska, E. (2008). Expressive power and abstraction in essence. *Constraints*, 13(3):343–384.

Mitchell, D. G., Ternovska, E., Hach, F., and Mohebali, R. (2006). Model expansion as a framework for modelling and solving search problems. Technical Report TR 2006-24, Simon Fraser University, Canada.

Niemelä, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273.

Nilsson, H. (1999). Tracing piece by piece: Affordable debugging for lazy functional languages. In Remy, D. and Lee, P., editors, *International Conference on Functional Programming (ICFP)*, pages 36–47.

Oberschelp, A. (1962). Untersuchungen zur mehrsortigen quantorenlogik. *Mathematische Annalen*, 145(4):297–333.

Oberschelp, A. (1989). Order sorted predicate logic. In Bläsius, K., Hedtstück, U., and Rollinger, C.-R., editors, *Sorts and Types in Artificial Intelligence*, pages 8–17. Springer-Verlag.

Patterson, M., Liu, Y., Ternovska, E., and Gupta, A. (2007). Grounding for model expansion in k-guarded formulas with inductive definitions. In Veloso, M. M., editor, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 161–166.

Pelov, N., Denecker, M., and Bruynooghe, M. (2007). Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming (TPLP)*, 7(3):301–353.

Pelov, N. and Ternovska, E. (2005). Reducing inductive definitions to propositional satisfiability. In Gabbrielli, M. and Gupta, G., editors, *International Conference on Logic Programming (ICLP)*, volume 3668 of *Lecture Notes in Computer Science*, pages 221–234. Springer.

Perri, S., Scarcello, F., Catalano, G., and Leone, N. (2007). Enhancing DLV instantiator by backjumping techniques. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):195–228.

Pettorossi, A. and Proietti, M. (1998). Transformation of logic programs. In Gabbay, D. M., Hogger, C. J., and Robinson, J. A., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 697–787. Oxford University Press.

Pipatsrisawat, K. and Darwiche, A. (2007). Clone: Solving weighted max-sat in a reduced search space. In Orgun, M. A. and Thornton, J., editors, *Australian Conference on Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 223–233. Springer.

Piper, J. (2000). *The Pleasures of God. Meditations on God's Delight in Being God*. Multnomah.

Pontelli, E. and Son, T. C. (2006). *Justifications* for logic programs under answer set semantics. In Etalle and Truszczyński (2006), pages 196–210.

Ramachandran, D. and Amir, E. (2005). Compact propositional encodings of first-order theories. In Veloso and Kambhampati (2005), pages 340–345.

Russell, B. (1905). On denoting. *Mind*, 14(56):479–493.

Schlipf, J. S. (1995). Complexity and undecidability results for logic programming. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):257–288.

Schulz, S. (2002). A comparison of different techniques for grounding near-propositional cnf formulae. In Haller, S. M. and Simmons, G., editors, *International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 72–76. AAAI Press.

Selman, B., Kautz, H., and Cohen, B. (1993). Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532.

Shapiro, E. Y. (1983). *Algorithmic Program DeBugging*. MIT Press, Cambridge, MA, USA.

Shlyakhter, I., Seater, R., Jackson, D., Sridharan, M., and Taghdiri, M. (2003a). Debugging overconstrained declarative models using unsatisfiable cores. In *International Conference on Automated Software Engineering (ASE)*, pages 94–105. IEEE Computer Society.

Shlyakhter, I., Sridharan, M., Seater, R., and Jackson, D. (2003b). Exploiting subformula sharing in automatic analysis of quantified formulas. Poster presented at Theory and Applications of Satisfiability Testing (SAT), 6th International Conference.

Simons, P., Niemelä, I., and Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234.

Sipser, M. (2005). *Introduction to the Theory of Computation*. Course Technology, 2nd edition.

Stuckenschmidt, H. (2007). Partial matchmaking using approximate subsumption. In Holte and Howe (2007), pages 1459–1464.

Sutcliffe, G. (2009). The TPTP problem library and associated infrastructure: The FOF and CNF parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362.

Swift, T. (2009). An engine for computing well-founded models. In Hill and Warren (2009), pages 514–518.

Syrjänen, T. (1998). Implementation of local grounding for logic programs with stable model semantics. Technical Report B18, Helsinki University of Technology, Finland.

Syrjänen, T. (2000). Lparse 1.0 user's manual. `http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz`.

Syrjänen, T. (2006). Debugging inconsistent answer set programs. In Dix, J. and Hunter, A., editors, *Workshop on Nonmonotonic Reasoning (NMR)*, pages 77–84.

Syrjänen, T. (2009). *Logic Programs and Cardinality Constraints: Theory and Practice*. Doctoral dissertation, TKK Dissertations in Information and Computer Science TKK-ICS-D12, Helsinki University of Technology, Faculty of Information and Natural Sciences, Department of Information and Computer Science, Espoo, Finland.

Ternovska, E. and Mitchell, D. G. (2009). Declarative programming of search problems with built-in arithmetic. In Boutilier, C., editor, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 942–947.

Torlak, E. and Jackson, D. (2007). Kodkod: A relational model finder. In Grumberg, O. and Huth, M., editors, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *Lecture Notes in Computer Science*, pages 632–647. Springer.

Tronçon, R. and Janssens, G. (2007). A delta debugger for ILP query execution. *CoRR*, abs/cs/0701105.

Tseitin, G. S. (1968). On the complexity of derivation in propositional calculus. In Slisenko, A. O., editor, *Studies in Constructive Mathematics and Mathematical Logic II*, volume 8 of *Seminars in Mathematics: Steklov Mathematical Institute*, pages 115–125. Consultants Bureau, New York.

Ullman, J. D. (1988). *Principles of database and knowledge-base systems, Vol. I.* Computer Science Press, Inc., New York, NY, USA.

van Fraassen, B. (1966). Singular terms, truth-value gaps and free logic. *Journal of Philosophy*, 63(17):481–495.

Van Gelder, A., Ross, K. A., and Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650.

Vardi, M. Y. (1982). The complexity of relational query languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146. ACM.

Veloso, M. M. and Kambhampati, S., editors (2005). *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. AAAI Press / The MIT Press.

Vennekens, J., Gilis, D., and Denecker, M. (2006). Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Transactions on Computational Logic*, 7(4):765–797.

Vennekens, J., Mariën, M., Wittocx, J., and Denecker, M. (2007). Predicate introduction for logics with a fixpoint semantics. Part I: Logic programming. *Fundamenta Informaticae*, 79(1-2):187–208.

Vlaeminck, H., Vennekens, J., and Denecker, M. (2009). A logical framework for configuration software. In Porto, A. and López-Fraguas, F. J., editors, *International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 141–148. ACM.

Wittocx, J., De Cat, B., and Denecker, M. (2009a). Towards computing revised models for FO theories. In Abreu, S. and Seipel, D., editors, *International Conference on Applications of Declarative Programming and Knowledge Management (INAP)*, pages 199–212.

Wittocx, J. and Mariën, M. (2008). The IDP system. `http://www.cs.kuleuven.be/~dtai/krr/software/idpmanual.pdf`.

Wittocx, J., Mariën, M., and Denecker, M. (2008a). Approximate reasoning in first-order logic theories. In Brewka and Lang (2008), pages 103–112.

Wittocx, J., Mariën, M., and Denecker, M. (2008b). GIDL: A grounder for FO^+. In Pagnucco, M. and Thielscher, M., editors, *Workshop on Nonmonotonic Reasoning (NMR)*, pages 189–198. University of New South Wales.

Wittocx, J., Mariën, M., and Denecker, M. (2008c). Grounding with bounds. In Fox and Gomes (2008), pages 572–577.

Wittocx, J., Mariën, M., and Denecker, M. (2008d). The IDP system: a model expansion system for an extension of classical logic. In *Workshop on Logic and Search (LaSh)*, pages 153–165.

Wittocx, J., Mariën, M., and Denecker, M. (2010). Grounding FO and FO(ID) with bounds. *Journal of Artificial Intelligence Research*. accepted.

Wittocx, J., Vlaeminck, H., and Denecker, M. (2009b). Debugging for model expansion. In Hill and Warren (2009), pages 296–311.

Index

Publication list

Articles in international reviewed journals

Joost Vennekens, Maarten Mariën, Johan Wittocx, and Marc Denecker. Predicate introduction for logics with a fixpoint semantics. Part I: Logic programming. *Fundamenta Informaticae*, 79(1-2):187–208, September 2007.

Joost Vennekens, Maarten Mariën, Johan Wittocx, and Marc Denecker. Predicate introduction for logics with a fixpoint semantics. Part II: Autoepistemic logic. *Fundamenta Informaticae*, 79(1-2):209–227, September 2007.

Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO(ID) with bounds. *Journal of Artificial Intelligence Research*, 2010. To appear.

Contributions at international conferences, published in proceedings (LNCS/LNAI or AAAI)

Ping Hou, Johan Wittocx, and Marc Denecker. A deductive system for PC(ID). In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, pages 162–174. Springer, 2007.

Maarten Mariën, Johan Wittocx, and Marc Denecker. Integrating inductive definitions in SAT. In Nachum Derschowitz and Andrei Voronkov, editors, *LPAR*, volume 4790 of *Lecture Notes in Computer Science*, pages 378–392. Springer, 2007.

Maarten Mariën, Johan Wittocx, Marc Denecker, and Maurice Bruynooghe. SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In Hans Kleine Büning and Xishun Zhao, editors, *SAT*, volume 4996 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2008.

Johan Wittocx, Joost Vennekens, Maarten Mariën, Marc Denecker, and Maurice Bruynooghe. Predicate introduction under stable and well-founded semantics. In Sandro Etalle and Mirosław Truszczyński, editors, *ICLP*, volume 4079 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2006.

Johan Wittocx, Maarten Mariën, and Marc Denecker. Approximate reasoning in first-order logic theories. In Gerhard Brewka and Jerôme Lang, editors, *KR*, pages 103–112. AAAI Press, 2008.

Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding with bounds. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 572–577. AAAI Press, 2008.

Johan Wittocx, Hanne Vlaeminck, and Marc Denecker. Debugging for model expansion. In Patricia M. Hill and David Scott Warren, editors, *ICLP*, volume 5649 of *Lecture Notes in Computer Science*, pages 296–311. Springer, 2009.

Contributions at international conferences, published elsewhere

Maarten Mariën, Johan Wittocx, and Marc Denecker. The IDP framework for declarative problem solving. In *Search and Logic: Answer Set Programming and SAT*, pages 19–34, 2006.

Maarten Mariën, Johan Wittocx, and Marc Denecker. MidL: a SAT(ID) solver. In *4th Workshop on Answer Set Programming: Advances in Theory and Implementation*, pages 303–308, 2007.

Johan Wittocx, Maarten Mariën, and Marc Denecker. GIDL: A grounder for FO^+. In Maurice Pagnucco and Michael Thielscher, editors, *NMR*, pages 189–198. University of New South Wales, 2008.

Johan Wittocx, Maarten Mariën, and Marc Denecker. The IDP system: a model expansion system for an extension of classical logic. In Marc Denecker, editor, *LaSh*, pages 153–165, 2008.

Johan Wittocx, Broes De Cat, and Marc Denecker. Towards computing revised models for FO theories. In Salvador Abreu and Dietmar Seipel, editors, *INAP*, pages 199–212, 2009.

Biography

Johan Wittocx was born on 6 June 1983 in Mechelen, Belgium. After finishing high school at the Berthoutinstituut - Klein Seminarie in Mechelen, he started studying Mathematics at the Katholieke Universiteit Leuven in 2001. In 2005, he received the Master's degree of Science in Mathematics (Licentiaat Wiskunde). His Master's thesis, titled *Decision problems and theorem provers*, was supervised by Professor Jan Denef.
Since September 2005, Johan has been working as a Ph.D. student at the DTAI research group of the Department of Computer Science at the Katholieke Universiteit Leuven. In October 2006, he obtained a personal Ph.D. grant from the Fonds voor Wetenschappelijk Onderzoek - Vlaanderen (FWO-Vlaanderen). His research was supervised by Professor Marc Denecker.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science
Informatics Section
Celestijnenlaan 200A box 2402
B-3001 Leuven